CENTRE FOR RAILWAY INFORMATION SYSTEMS

RAIL INDEX PREDICTION

INTRODUCTION

In this project, we worked on oscillation parameter range preparation using real inspection data collected from Indian Railways. The parameters of interest, VRI (Vertical Rail Index) and LRI (Lateral Rail Index), were analysed to generate valid date ranges during which a specific value remained applicable. The purpose is to track how these oscillation parameters change over time for each railway block section.

DATASET DESCRIPTION

- **File Name:** OMSRI_2014_2018.csv
- **Source:** Oscillation Management System, Indian Railways
- **Time Period:** 2014 to 2018
- **Main Columns Used:**
    - RUNDATE: Timestamp of inspection
    - SECCODE: Section code
    - LINECODE: Line code
    - KMFROM: Kilometres from starting point
    - BLOCKNO: Block number
    - VRI, LRI: Oscillation parameters
- **Columns Dropped:** INSPID, SNAME, LINE, RUNNO (not relevant for analysis)

OBJECTIVE

The primary objective of this project is to transform point-based inspection data (from each inspection date) into a continuous time-based range dataset for each oscillation parameter (VRI, LRI), where:

- Each row represents a date range (DATE1 to DATE2) for which a specific value of the parameter is applicable.
- If there are missing dates between inspections, we fill those gaps by carrying forward the last known parameter value.

METHODOLOGY

a. Data Loading & Cleaning

- Loaded the CSV file using pandas.
- Converted the RUNDATE string to a datetime format.
- Removed irrelevant columns and rows with missing essential values (RUNDATE, SECCODE, BLOCKNO, etc.).

b. Data Reshaping (Long Format)

- Used pd.melt() to convert VRI and LRI columns into a single PARAM column with corresponding values in INITIAL_PAR.
- Sorted the data to prepare for range generation.

c. Time Range Generation

- Created DATE1 and DATE2 for each row:
- DATE1 is the current inspection date.
- DATE2 is the next inspection date (shifted).
- Created FINAL_PAR to hold the next parameter value.

d. Gap Filling

- Checked for missing date ranges between DATE2 and next DATE1.
- Filled gaps with a new row carrying forward the last known value (i.e., INITIAL_PAR = FINAL_PAR).
- Ensured no loss of continuity in time ranges.

e. Final Formatting

- Combined the original and gap-filled data.
- Sorted and formatted date fields (DATE1, DATE2) to dd-mm-yyyy.
- Selected final columns:
  PARAM, SECCODE, LINECODE, KMFROM, BLOCKNO, DATE1, DATE2, INITIAL_PAR, FINAL_PAR.

RESULT

- A **final DataFrame** was generated that accurately represents the **value range** for VRI and LRI over time for each railway block section.
- Each row in the final output indicates the period during which a particular parameter value was valid.
- Gaps in data were filled to ensure **continuous coverage** across the timeline.
- The result is suitable for:
- Visualization of parameter evolution over time.
- Further analysis like threshold detection, oscillation alerts, or model training.

CODE SNIPPET

```
import pandas as pd

df = pd.read_csv(r'D:\Kavin\Intership-CRIS\Osicillation Management
System\OMSRI_2014_2018.csv')

pd.set_option('Display.max_columns',None)

df['RUNDATE'] = pd.to_datetime(df['RUNDATE'], format='%Y-%m-%d-%H.%M.%S',
errors='coerce')

df.drop(columns=['INSPID', 'SNAME', 'LINE', 'RUNNO'], errors='ignore', inplace=True)

df.dropna(subset=['RUNDATE', 'SECCODE', 'LINECODE', 'KMFROM', 'BLOCKNO',
'VRI', 'LRI'], inplace=True)

df_long = df.melt(
    id_vars=['SECCODE', 'LINECODE', 'KMFROM', 'BLOCKNO', 'RUNDATE'],
    value_vars=['VRI', 'LRI'],
    var_name='PARAM',
    value_name='INITIAL_PAR'
)

df_long.sort_values(by=['SECCODE', 'LINECODE', 'KMFROM', 'BLOCKNO', 'PARAM',
'RUNDATE'], inplace=True)
df_long['DATE1'] = df_long['RUNDATE']
df_long['DATE2'] = df_long.groupby(['SECCODE', 'LINECODE', 'KMFROM',
'BLOCKNO', 'PARAM'])['DATE1'].shift(-1)
df_long['FINAL_PAR'] = df_long.groupby(['SECCODE', 'LINECODE', 'KMFROM',
'BLOCKNO', 'PARAM'])['INITIAL_PAR'].shift(-1)
df_long.drop(columns=['RUNDATE'], inplace=True)
df_long = df_long.dropna(subset=['DATE2'])

filled_rows = []
for keys, group in df_long.groupby(['SECCODE', 'LINECODE', 'KMFROM', 'BLOCKNO',
'PARAM']):
    group = group.sort_values('DATE1').reset_index(drop=True)
    for i in range(len(group) - 1):
        curr = group.loc[i]
        next_row = group.loc[i + 1]
        if curr['DATE2'].date() < next_row['DATE1'].date():
            gap = curr.copy()
            gap['DATE1'] = curr['DATE2']
            gap['DATE2'] = next_row['DATE1']
            gap['INITIAL_PAR'] = curr['FINAL_PAR']
            gap['FINAL_PAR'] = curr['FINAL_PAR']
            filled_rows.append(gap)
    filled_rows.append(group.loc[len(group)-1])
```

```
gap_df = pd.DataFrame(filled_rows)
combined = pd.concat([df_long, gap_df], ignore_index=True)
combined.sort_values(by=['SECCODE', 'LINECODE', 'KMFROM', 'BLOCKNO', 'DATE1',
'PARAM'], inplace=True)
combined['DATE1'] = combined['DATE1'].dt.strftime('%d-%m-%Y')
combined['DATE2'] = combined['DATE2'].dt.strftime('%d-%m-%Y')

final_df = combined[['PARAM', 'SECCODE', 'LINECODE', 'KMFROM', 'BLOCKNO',
'DATE1', 'DATE2', 'INITIAL_PAR', 'FINAL_PAR']]
final_df.reset_index(drop=True, inplace=True)

print(final_df.head(6))

#final_df.to_csv(r'D:\Kavin\Intership-CRIS\OP_2019-2025.csv',index=False)
```

OUTPUT

```
In [2]: %runfile 'D:/Kavin/Intership-CRIS/Osicillation Management System/
oms_1.py' --wdir
  PARAM  SECCODE  LINECODE  KMFROM  BLOCKNO       DATE1       DATE2  \
0   LRI        1       1.0  1333.0      1.0  03-01-2014  24-02-2014
1   VRI        1       1.0  1333.0      1.0  03-01-2014  24-02-2014
2   LRI        1       1.0  1333.0      1.0  24-02-2014  21-03-2014
3   VRI        1       1.0  1333.0      1.0  24-02-2014  21-03-2014
4   LRI        1       1.0  1333.0      1.0  21-03-2014  19-05-2014
5   VRI        1       1.0  1333.0      1.0  21-03-2014  19-05-2014

   INITIAL_PAR  FINAL_PAR
0         2.57       2.64
1         1.86       2.56
2         2.64       3.12
3         2.56       2.71
4         3.12       2.82
5         2.71       2.55
```

INTRODUCTION

Here we worked on developing a machine learning model aimed at predicting rail index in trains using data related to the Oscillation Management System Rail Index (OMSRI). The objective of the project was to build and optimize a predictive model for rail index prediction.

DATASET DESCRIPTION

The dataset provided by CRIS contained various parameters collected from OMSI reports, including but not limited to:

- SECCODE: Section code
- LINECODE: Line identifier
- KMFROM: Starting kilometer
- BLOCKNO: Block number
- PARAM: Type of parameter measured
- GMT: Timestamp or time reference

Preprocessing steps involved handling missing values, encoding categorical features, feature scaling where applicable, and train-test splitting (typically 70%-30%).

Machine Learning Model

Model Used

Random Forest Regressor:
Random Forest builds multiple decision trees and averages their predictions, making it robust to overfitting and effective for capturing non-linear patterns.

XGBoost Regressor:
XGBoost is a fast and efficient gradient boosting algorithm that offers strong performance through regularization, tree pruning, and handling of missing values.

LightGBM Regressor:
LightGBM uses histogram-based techniques for fast training and low memory usage. It excels with large datasets and high-dimensional features, and natively supports categorical data.

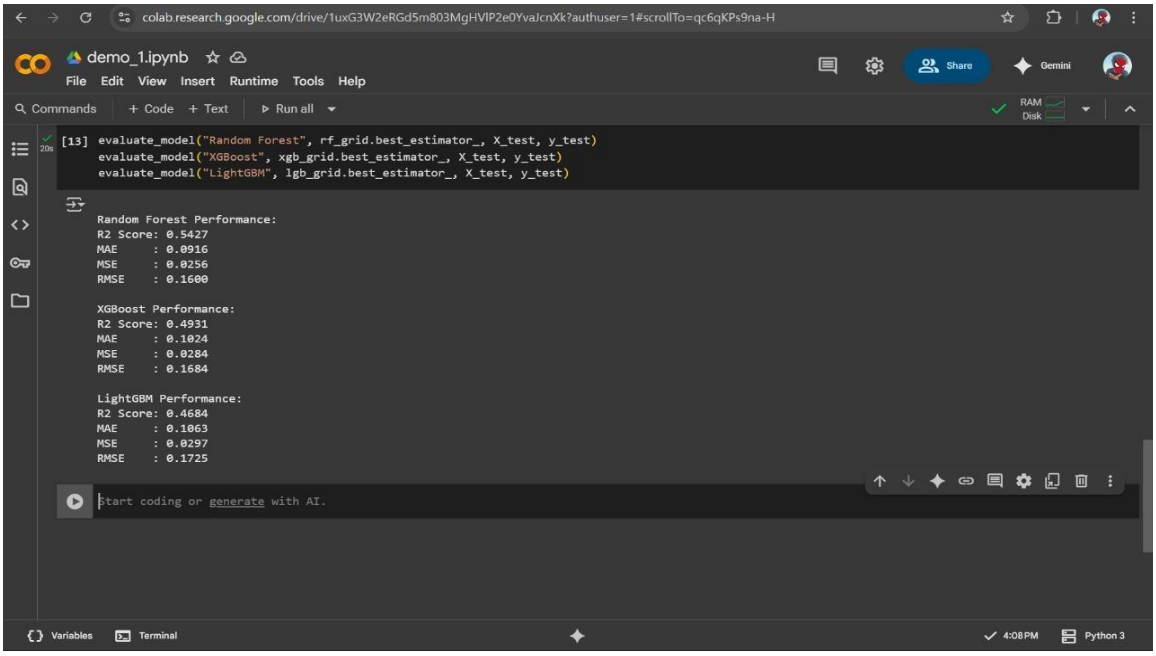
INITIAL MODEL CODE SNIPPET

```
!pip install xgboost lightgbm scikit-learn --quiet

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('/content/drive/My Drive/your_path/training_data.csv')  # <-- Update path

label_cols = ['SECCODE', 'LINECODE', 'BLOCKNO', 'PARAM', 'GMT']
le = LabelEncoder()
for col in label_cols:
    df[col] = le.fit_transform(df[col].astype(str))

X = df[["SECCODE", "LINECODE", "KMFROM", "BLOCKNO", "PARAM", "RI1",
"GMT"]]
y = df["RI2"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

rf_params = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
}
xgb_params = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.1, 0.2],
}
lgb_params = {
    'n_estimators': [50, 100, 150],
    'max_depth': [5, 10, -1],
    'learning_rate': [0.01, 0.1],
}

rf_search = RandomizedSearchCV(RandomForestRegressor(random_state=42), rf_params,
n_iter=5, cv=3, scoring='r2', n_jobs=-1, random_state=42)
xgb_search = RandomizedSearchCV(XGBRegressor(random_state=42, verbosity=0),
xgb_params, n_iter=5, cv=3, scoring='r2', n_jobs=-1, random_state=42)
lgb_search = RandomizedSearchCV(LGBMRegressor(random_state=42), lgb_params,
n_iter=5, cv=3, scoring='r2', n_jobs=-1, random_state=42)
```

```
rf_search.fit(X_train, y_train)
xgb_search.fit(X_train, y_train)
lgb_search.fit(X_train, y_train)

def evaluate_model(name, model):
    y_pred = model.predict(X_test)
    print(f"\n {name} Results:")
    print("Best Parameters:", model.best_params_)
    print("R2 Score:", r2_score(y_test, y_pred))
    print("MAE:", mean_absolute_error(y_test, y_pred))
    print("MSE:", mean_squared_error(y_test, y_pred))
    print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))

evaluate_model("Random Forest", rf_search)
evaluate_model("XGBoost", xgb_search)
evaluate_model("LightGBM", lgb_search)
```

INITIAL MODEL RESULTS

- R2 Score: 50.014 %
- RMSE: 16.669 %



HYPERPARAMETER TUNING

TUNING CODE SNIPPET

```
!pip install pandas scikit-learn xgboost lightgbm --quiet

  import pandas as pd
  import numpy as np
  from sklearn.model_selection import train_test_split, RandomizedSearchCV
  from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
  vri_df = pd.read_csv(r"/content/drive/MyDrive/Intern-CRIS/training_data_vri.csv")
  lri_df = pd.read_csv(r"/content/drive/MyDrive/Intern-CRIS/training_data_lri.csv")
  df = pd.concat([vri_df, lri_df], ignore_index=True).dropna()
  X = df[["LINECODE", "KMFROM", "PARAM", "RI1", "GMT"]]
  y = df["RI2"]
  X = pd.get_dummies(X, drop_first=True)

  X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.3, random_state=42
)
  lgb_params = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, 15],
    'learning_rate': [0.05, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}
lgb_search = RandomizedSearchCV(
    LGBMRegressor(random_state=42),
    param_distributions=lgb_params,
    n_iter=10,
    scoring='r2',
    cv=3,
    random_state=42,
    n_jobs=-1
)
lgb_search.fit(X_train, y_train)
def evaluate(model, X_test, y_test):
    preds = model.predict(X_test)
    print(" Optimized LightGBM Results:")
    print(f"R² Score: {r2_score(y_test, preds):.4f}")
    print(f"MAE      : {mean_absolute_error(y_test, preds):.4f}")
    print(f"MSE      : {mean_squared_error(y_test, preds):.4f}")
    print(f"RMSE     : {np.sqrt(mean_squared_error(y_test, preds)):.4f}")
    print(f"Best Params: {model.get_params()}")

evaluate(lgb_search.best_estimator_, X_test, y_test)
xgb_model = XGBRegressor(random_state=42)
xgb_model.fit(X_train, y_train)
```

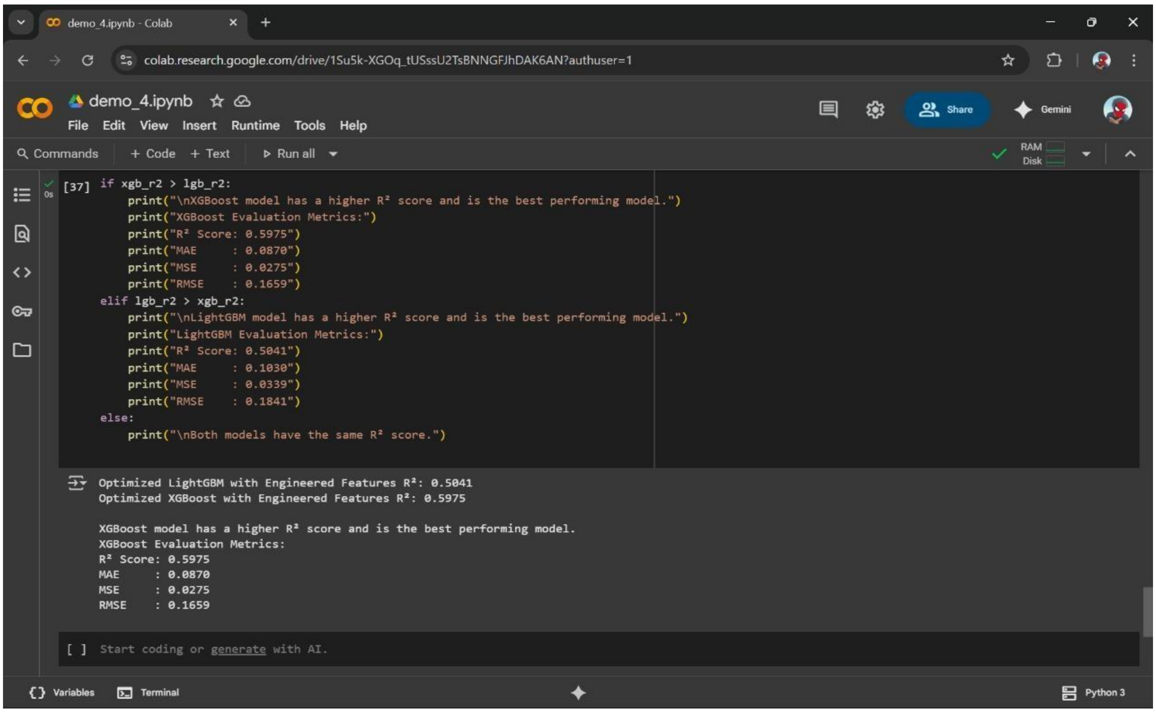
```
print("XGBoost Results:")
evaluate(xgb_model, X_test, y_test)

from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
print("\n Linear Regression Results:")
evaluate(lr_model, X_test, y_test)

from sklearn.linear_model import Ridge

ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
print("\n Ridge Regression Results:")
evaluate(ridge_model, X_test, y_test)
xgb_params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 6, 9, 12],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'subsample': [0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0]
}

xgb_search = RandomizedSearchCV(
    XGBRegressor(random_state=42),
    param_distributions=xgb_params,
    n_iter=20,
    scoring='r2',
    cv=3,
    random_state=42,
    n_jobs=-1
)

xgb_search.fit(X_train, y_train)

print("\n Optimized XGBoost Results:")
evaluate(xgb_search.best_estimator_, X_test, y_test)
df['DATE1'] = pd.to_datetime(df['DATE1'])
df['DATE2'] = pd.to_datetime(df['DATE2'])
df['TIME_DIFF'] = (df['DATE2'] - df['DATE1']).dt.days
df['DATE1_YEAR'] = df['DATE1'].dt.year
df['DATE1_MONTH'] = df['DATE1'].dt.month
df['DATE1_DAY'] = df['DATE1'].dt.day
df['DATE2_YEAR'] = df['DATE2'].dt.year
df['DATE2_MONTH'] = df['DATE2'].dt.month
df['DATE2_DAY'] = df['DATE2'].dt.day

df['RI1_GMT_INTERACTION'] = df['RI1'] * df['GMT']
df['RI1_squared'] = df['RI1']**2
```

```
X = df[["LINECODE", "KMFROM", "PARAM", "RI1", "GMT", "TIME_DIFF", "DATE1_YEAR", "DATE1_MONTH", "DATE1_DAY", "DATE2_YEAR", "DATE2_MONTH", "DATE2_DAY", "RI1_GMT_INTERACTION", "RI1_squared"]]
y = df["RI2"]

X = pd.get_dummies(X, drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
lgb_preds = lgb_search.best_estimator_.predict(X_test)

print("Optimized LightGBM Results with Engineered Features:")
print(f"R² Score: {r2_score(y_test, lgb_preds):.4f}")
print(f"MAE     : {mean_absolute_error(y_test, lgb_preds):.4f}")
print(f"MSE     : {mean_squared_error(y_test, lgb_preds):.4f}")
print(f"RMSE    : {np.sqrt(mean_squared_error(y_test, lgb_preds)):.4f}")
best_lgb_model = LGBMRegressor(**lgb_search.best_params_, random_state=42)
best_lgb_model.fit(X_train, y_train)

lgb_preds = best_lgb_model.predict(X_test)

print("Optimized LightGBM Results with Engineered Features (Retrained):")
print(f"R² Score: {r2_score(y_test, lgb_preds):.4f}")
print(f"MAE     : {mean_absolute_error(y_test, lgb_preds):.4f}")
print(f"MSE     : {mean_squared_error(y_test, lgb_preds):.4f}")
print(f"RMSE    : {np.sqrt(mean_squared_error(y_test, lgb_preds)):.4f}")
best_xgb_model = XGBRegressor(**xgb_search.best_params_, random_state=42)
best_xgb_model.fit(X_train, y_train)

xgb_preds = best_xgb_model.predict(X_test)

print(" Optimized XGBoost Results with Engineered Features (Retrained):")
print(f"R² Score: {r2_score(y_test, xgb_preds):.4f}")
print(f"MAE     : {mean_absolute_error(y_test, xgb_preds):.4f}")
print(f"MSE     : {mean_squared_error(y_test, xgb_preds):.4f}")
print(f"RMSE    : {np.sqrt(mean_squared_error(y_test, xgb_preds)):.4f}")
lgb_r2 = 0.5041
xgb_r2 = 0.5975

print(f"Optimized LightGBM with Engineered Features R²: {lgb_r2:.4f}")
print(f"Optimized XGBoost with Engineered Features R²: {xgb_r2:.4f}")

if xgb_r2 > lgb_r2:
    print("\nXGBoost model has a higher R² score and is the best performing model.")
    print("XGBoost Evaluation Metrics:")
    print("R² Score: 0.5975")
    print("MAE     : 0.0870")
    print("MSE     : 0.0275")
```

```
    print("RMSE    : 0.1659")
elif lgb_r2 > xgb_r2:
    print("\nLightGBM model has a higher R² score and is the best performing model.")
    print("LightGBM Evaluation Metrics:")
    print("R² Score: 0.5041")
    print("MAE     : 0.1030")
    print("MSE     : 0.0339")
    print("RMSE    : 0.1841")
else:
    print("\nBoth models have the same R² score.")
```

TUNING RESULTS

- Tuned R2 Score: 59.75 %
- Tuned RMSE: 16.59 %



MODEL TRAINING WITH FILTERED DATASET

To enhance the model's accuracy and reliability, a filtering condition was applied to the dataset: only records where RI2 > RI1 were considered. This condition ensures the model learns from scenarios where there is a meaningful increase in the oscillation index, which could be critical for identifying problematic track sections.

After filtering, the selected features for training were:

- **X (Independent Variables):** ["LINECODE", "SECCODE", "BLOCKNO", "KMFROM", "PARAM", "RI1", "GMT"]
- **y (Target Variable):** ["RI2"]

The model was retrained on this refined dataset, and its performance was evaluated using appropriate regression metrics.

CODE SNIPPET: MODEL TRAINING WITH FILTERED RI2 > RI1 DATASET:

```
X = pd.get_dummies(X, drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.3, random_state=42
)

def objective(trial):
xgb_params = {
'n_estimators': trial.suggest_int('n_estimators', 100, 1500),
'max_depth': trial.suggest_int('max_depth', 3, 20),
'learning_rate': trial.suggest_float('learning_rate', 0.005, 0.5, log=True),
'subsample': trial.suggest_float('subsample', 0.5, 1.0),
'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1.0),
'gamma': trial.suggest_float('gamma', 0, 1.0),
'reg_alpha': trial.suggest_float('reg_alpha', 0, 1.0),
'reg_lambda': trial.suggest_float('reg_lambda', 0, 1.0),
'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
'colsample_bylevel': trial.suggest_float('colsample_bylevel', 0.5, 1.0),
'random_state': 42,
'n_jobs': -1
}
model = XGBRegressor(**xgb_params)
scores = cross_val_score(model, X_train, y_train, cv=5, scoring='r2', n_jobs=-1)
return scores.mean()

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

print("\n Optuna optimization finished.")
print("Best hyperparameters:", study.best_params)
```

```
print(f"Best R² score (cross-validated): {study.best_value:.4f}")

best_xgb_model = XGBRegressor(**study.best_params, random_state=42, n_jobs=-1)
best_xgb_model.fit(X_train, y_train)

print("\n Optimized XGBoost Results with Engineered Features (Retrained):")
print(f"R² Score: {r2_score(y_test, best_xgb_model.predict(X_test)):.4f}")
print(f"MAE : {mean_absolute_error(y_test, best_xgb_model.predict(X_test)):.4f}")
print(f"MSE : {mean_squared_error(y_test, best_xgb_model.predict(X_test)):.4f}")
print(f"RMSE : {np.sqrt(mean_squared_error(y_test, best_xgb_model.predict(X_test))):.4f}
```

FILTERED MODEL RESULTS

- Tuned R2 Score: 79.86 %
- Tuned RMSE: 9.02 %

```
[ ] xgb_optuna_preds = best_xgb_model_optuna.predict(X_test)

    print("\n✅ Optimized XGBoost Results (Optuna) on Test Set:")
    print(f"R² Score: {r2_score(y_test, xgb_optuna_preds):.4f}")
    print(f"MAE     : {mean_absolute_error(y_test, xgb_optuna_preds):.4f}")
    print(f"MSE     : {mean_squared_error(y_test, xgb_optuna_preds):.4f}")
    print(f"RMSE    : {np.sqrt(mean_squared_error(y_test, xgb_optuna_preds)):.4f}")

    ✅ Optimized XGBoost Results (Optuna) on Test Set:
    R² Score: 0.7986
    MAE     : 0.0621
    MSE     : 0.0081
    RMSE    : 0.0902
```

CODE SNIPPET: MODEL TRAINING WITH XGBOOST AND LIGHTGBM AND ENSEMBLED USING RIDGE CV:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from xgboost import XGBRegressor
import lightgbm as lgb
from sklearn.linear_model import RidgeCV

vri_df = pd.read_csv(r"/content/drive/MyDrive/Intern-CRIS/Dataset/training_data_vri.csv")
lri_df = pd.read_csv(r"/content/drive/MyDrive/Intern-CRIS/Dataset/training_data_lri.csv")

df = pd.concat([vri_df, lri_df], ignore_index=True).dropna()
df = df[df['RI2'] > df['RI1']]

df['DATE1'] = pd.to_datetime(df['DATE1'])
df['DATE2'] = pd.to_datetime(df['DATE2'])

df['TIME_DIFF'] = (df['DATE2'] - df['DATE1']).dt.days

df['DATE1_YEAR'] = df['DATE1'].dt.year
df['DATE1_MONTH'] = df['DATE1'].dt.month
df['DATE1_DAY'] = df['DATE1'].dt.day
df['DATE2_YEAR'] = df['DATE2'].dt.year
df['DATE2_MONTH'] = df['DATE2'].dt.month
df['DATE2_DAY'] = df['DATE2'].dt.day

df['DATE2_YEAR_RI1_INTERACTION'] = df['DATE2_YEAR'] * df['RI1']
df['TIME_DIFF_RI1_INTERACTION'] = df['TIME_DIFF'] * df['RI1']
df['DATE2_YEAR_TIME_DIFF_INTERACTION'] = df['DATE2_YEAR'] *
df['TIME_DIFF']
df['DATE2_MONTH_RI1_INTERACTION'] = df['DATE2_MONTH'] * df['RI1']

df['RI1_squared'] = df['RI1'] ** 2
df['GMT_squared'] = df['GMT'] ** 2
df['TIME_DIFF_squared'] = df['TIME_DIFF'] ** 2

df['RI1_rolling_avg'] = df.groupby(['LINECODE', 'SECCODE'])['RI1'].transform(lambda x:
x.rolling(window=5, min_periods=1).mean())
df['RI1_diff_from_avg'] = df['RI1'] - df['RI1_rolling_avg']

df['GMT_DATE1_MONTH_INTERACTION'] = df['GMT'] * df['DATE1_MONTH']
df['GMT_DATE2_MONTH_INTERACTION'] = df['GMT'] * df['DATE2_MONTH']
df['GMT_DATE1_YEAR_INTERACTION'] = df['GMT'] * df['DATE1_YEAR']
df['GMT_DATE2_YEAR_INTERACTION'] = df['GMT'] * df['DATE2_YEAR']

df['RI1_cubed'] = df['RI1'] ** 3
```

```
X = df[["LINECODE", "SECCODE", "BLOCKNO", "KMFROM", "PARAM", "RI1",
"GMT",
        "TIME_DIFF", "DATE1_YEAR", "DATE1_MONTH", "DATE1_DAY",
"DATE2_YEAR",
        "DATE2_MONTH", "DATE2_DAY", "DATE2_YEAR_RI1_INTERACTION",
        "TIME_DIFF_RI1_INTERACTION",
"DATE2_YEAR_TIME_DIFF_INTERACTION",
        "DATE2_MONTH_RI1_INTERACTION", "RI1_squared", "GMT_squared",
        "TIME_DIFF_squared", 'RI1_diff_from_avg',
'GMT_DATE1_MONTH_INTERACTION',
        'GMT_DATE2_MONTH_INTERACTION', 'GMT_DATE1_YEAR_INTERACTION',
        'GMT_DATE2_YEAR_INTERACTION', 'RI1_cubed']]

y = df["RI2"]

X = pd.get_dummies(X, drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

best_params = {
    'n_estimators': 1066,
    'learning_rate': 0.033844909001396695,
    'max_depth': 11,
    'subsample': 0.9833761615307697,
    'colsample_bytree': 0.8737870597207275,
    'gamma': 0.001008631417380501,
    'reg_alpha': 0.25412571901286984,
    'reg_lambda': 0.7077459432486992,
    'min_child_weight': 5,
    'colsample_bylevel': 0.8881074179470817,
    'random_state': 42,
    'n_jobs': -1,
    'verbosity': 0
}

final_xgb_model = XGBRegressor(**best_params)
final_xgb_model.fit(X_train, y_train)

xgb_preds = final_xgb_model.predict(X_test)

print("\nFinal XGBoost Results with Initial Optuna Parameters and New Features:")
print(f"R² Score : {r2_score(y_test, xgb_preds):.4f}")
print(f"MAE      : {mean_absolute_error(y_test, xgb_preds):.4f}")
print(f"MSE      : {mean_squared_error(y_test, xgb_preds):.4f}")
print(f"RMSE     : {np.sqrt(mean_squared_error(y_test, xgb_preds)):.4f}")

best_lgbm_params = {
```

```
    'n_estimators': 1232,
    'learning_rate': 0.08527669010590563,
    'num_leaves': 213,
    'max_depth': 9,
    'min_child_samples': 24,
    'subsample': 0.6063446939538922,
    'colsample_bytree': 0.6302332455777844,
    'reg_alpha': 0.8092809147554415,
    'reg_lambda': 0.2513376320261217,
    'objective': 'regression',
    'random_state': 42,
    'n_jobs': -1,
    'verbosity': -1
}

final_lgbm_model = lgb.LGBMRegressor(**best_lgbm_params)
final_lgbm_model.fit(X_train, y_train)

lgbm_preds = final_lgbm_model.predict(X_test)

print("\nFinal LightGBM Results with Initial Optuna Parameters and New Features:")
print(f"R² Score : {r2_score(y_test, lgbm_preds):.4f}")
print(f"MAE      : {mean_absolute_error(y_test, lgbm_preds):.4f}")
print(f"MSE      : {mean_squared_error(y_test, lgbm_preds):.4f}")
print(f"RMSE     : {np.sqrt(mean_squared_error(y_test, lgbm_preds)):.4f}")

ensemble_preds_weighted = 0.5 * lgbm_preds + 0.5 * xgb_preds

print("Weighted Ensemble (50% LGBM + 50% XGB):")
print(f"R² Score : {r2_score(y_test, ensemble_preds_weighted):.4f}")

meta_X = pd.DataFrame({
    'xgb': xgb_preds,
    'lgbm': lgbm_preds
})

meta_model = RidgeCV()
meta_model.fit(meta_X, y_test)
stacked_preds = meta_model.predict(meta_X)

print("\nStacking with RidgeCV:")
print(f"R² Score : {r2_score(y_test, stacked_preds):.4f}")
```

MODEL RESULTS

- Tuned R2 Score: 81.02 %

```
from sklearn.linear_model import RidgeCV

meta_X = pd.DataFrame({
    'xgb': xgb_preds,
    'lgbm': lgbm_preds
})

meta_model = RidgeCV()
meta_model.fit(meta_X, y_test)
stacked_preds = meta_model.predict(meta_X)

print("\n✅ Stacking with RidgeCV:")
print(f"R² Score : {r2_score(y_test, stacked_preds):.4f}")
```

```
✅ Stacking with RidgeCV:
R² Score : 0.8102
```

CODE SNIPPET: MODEL TRAINING WITH XGBOOST AND LIGHTGBM AND ENSEMBLED USING RIDGE CV:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from xgboost import XGBRegressor
import lightgbm as lgb
from sklearn.ensemble import RandomForestRegressor

vri_df = pd.read_csv(r"/content/drive/MyDrive/Intern-CRIS/Dataset/training_data_vri.csv")
lri_df = pd.read_csv(r"/content/drive/MyDrive/Intern-CRIS/Dataset/training_data_lri.csv")

df = pd.concat([vri_df, lri_df], ignore_index=True).dropna()
df = df[df['RI2'] > df['RI1']]

df['DATE1'] = pd.to_datetime(df['DATE1'])
df['DATE2'] = pd.to_datetime(df['DATE2'])

df['TIME_DIFF'] = (df['DATE2'] - df['DATE1']).dt.days

df['DATE1_YEAR'] = df['DATE1'].dt.year
df['DATE1_MONTH'] = df['DATE1'].dt.month
df['DATE1_DAY'] = df['DATE1'].dt.day
df['DATE2_YEAR'] = df['DATE2'].dt.year
df['DATE2_MONTH'] = df['DATE2'].dt.month
df['DATE2_DAY'] = df['DATE2'].dt.day

df['DATE2_YEAR_RI1_INTERACTION'] = df['DATE2_YEAR'] * df['RI1']
df['TIME_DIFF_RI1_INTERACTION'] = df['TIME_DIFF'] * df['RI1']
df['DATE2_YEAR_TIME_DIFF_INTERACTION'] = df['DATE2_YEAR'] *
```

```
df['TIME_DIFF']
df['DATE2_MONTH_RI1_INTERACTION'] = df['DATE2_MONTH'] * df['RI1']

df['RI1_squared'] = df['RI1'] ** 2
df['GMT_squared'] = df['GMT'] ** 2
df['TIME_DIFF_squared'] = df['TIME_DIFF'] ** 2

df['RI1_rolling_avg'] = df.groupby(['LINECODE', 'SECCODE'])['RI1'].transform(lambda x:
x.rolling(window=5, min_periods=1).mean())
df['RI1_diff_from_avg'] = df['RI1'] - df['RI1_rolling_avg']

df['GMT_DATE1_MONTH_INTERACTION'] = df['GMT'] * df['DATE1_MONTH']
df['GMT_DATE2_MONTH_INTERACTION'] = df['GMT'] * df['DATE2_MONTH']
df['GMT_DATE1_YEAR_INTERACTION'] = df['GMT'] * df['DATE1_YEAR']
df['GMT_DATE2_YEAR_INTERACTION'] = df['GMT'] * df['DATE2_YEAR']

df['RI1_cubed'] = df['RI1'] ** 3



X = df[["LINECODE", "SECCODE", "BLOCKNO", "KMFROM", "PARAM", "RI1",
"GMT",
    "TIME_DIFF", "DATE1_YEAR", "DATE1_MONTH", "DATE1_DAY",
"DATE2_YEAR",
    "DATE2_MONTH", "DATE2_DAY", "DATE2_YEAR_RI1_INTERACTION",
    "TIME_DIFF_RI1_INTERACTION",
"DATE2_YEAR_TIME_DIFF_INTERACTION",
    "DATE2_MONTH_RI1_INTERACTION", "RI1_squared", "GMT_squared",
    "TIME_DIFF_squared", 'RI1_diff_from_avg',
'GMT_DATE1_MONTH_INTERACTION',
    'GMT_DATE2_MONTH_INTERACTION', 'GMT_DATE1_YEAR_INTERACTION',
    'GMT_DATE2_YEAR_INTERACTION', 'RI1_cubed']]
```

```
y = df["RI2"]

X = pd.get_dummies(X, drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

best_params = {
    'n_estimators': 1066,
    'learning_rate': 0.033844909001396695,
    'max_depth': 11,
    'subsample': 0.9833761615307697,
    'colsample_bytree': 0.8737870597207275,
    'gamma': 0.001008631417380501,
    'reg_alpha': 0.25412571901286984,
    'reg_lambda': 0.7077459432486992,
    'min_child_weight': 5,
    'colsample_bylevel': 0.8881074179470817,
    'random_state': 42,
    'n_jobs': -1,
    'verbosity': 0
}

final_xgb_model = XGBRegressor(**best_params)
final_xgb_model.fit(X_train, y_train)

xgb_preds = final_xgb_model.predict(X_test)

best_lgbm_params = {
    'n_estimators': 1647,
```

```
    'learning_rate': 0.046186029765235975,
    'num_leaves': 239,
    'max_depth': 14,
    'min_child_samples': 22,
    'subsample': 0.8047364810078328,
    'colsample_bytree': 0.6571494020123966,
    'reg_alpha': 0.9907226421605784,
    'reg_lambda': 0.39465333866606117,
    'objective': 'regression',
    'random_state': 42,
    'n_jobs': -1,
    'verbosity': -1
}

final_lgbm_model = lgb.LGBMRegressor(**best_lgbm_params)
final_lgbm_model.fit(X_train, y_train)

lgbm_preds = final_lgbm_model.predict(X_test)

meta_X_rf = pd.DataFrame({
    "xgb": xgb_preds,
    "lgbm": lgbm_preds
})

rf_meta_model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf_meta_model.fit(meta_X_rf, y_test)

stacked_preds_rf = rf_meta_model.predict(meta_X_rf)

print("Stacking Results (Meta-model: Random Forest Regressor):")
print(f"R² Score : {r2_score(y_test, stacked_preds_rf):.4f}")
```

print(f"MAE   : {mean_absolute_error(y_test, stacked_preds_rf):.4f}")

print(f"MSE   : {mean_squared_error(y_test, stacked_preds_rf):.4f}")

print(f"RMSE   : {np.sqrt(mean_squared_error(y_test, stacked_preds_rf)):.4f}")

MODEL RESULTS

- Tuned R2 Score: 96.89 %

```
print("\nStacking Results (Meta-model: Random Forest Regressor):")
print(f"R² Score : {r2_score(y_test, stacked_preds_rf):.4f}")
print(f"MAE      : {mean_absolute_error(y_test, stacked_preds_rf):.4f}")
print(f"MSE      : {mean_squared_error(y_test, stacked_preds_rf):.4f}")
print(f"RMSE     : {np.sqrt(mean_squared_error(y_test, stacked_preds_rf)):.4f}")
```

```
Stacking Results (Meta-model: Random Forest Regressor):
R² Score : 0.9689
MAE      : 0.0245
MSE      : 0.0013
RMSE     : 0.0355
```

JEEVESHRAJ D
SSN COLLEGE OF ENGG.

KAVIN RAAM M
SSN COLLEGE OF ENGG.