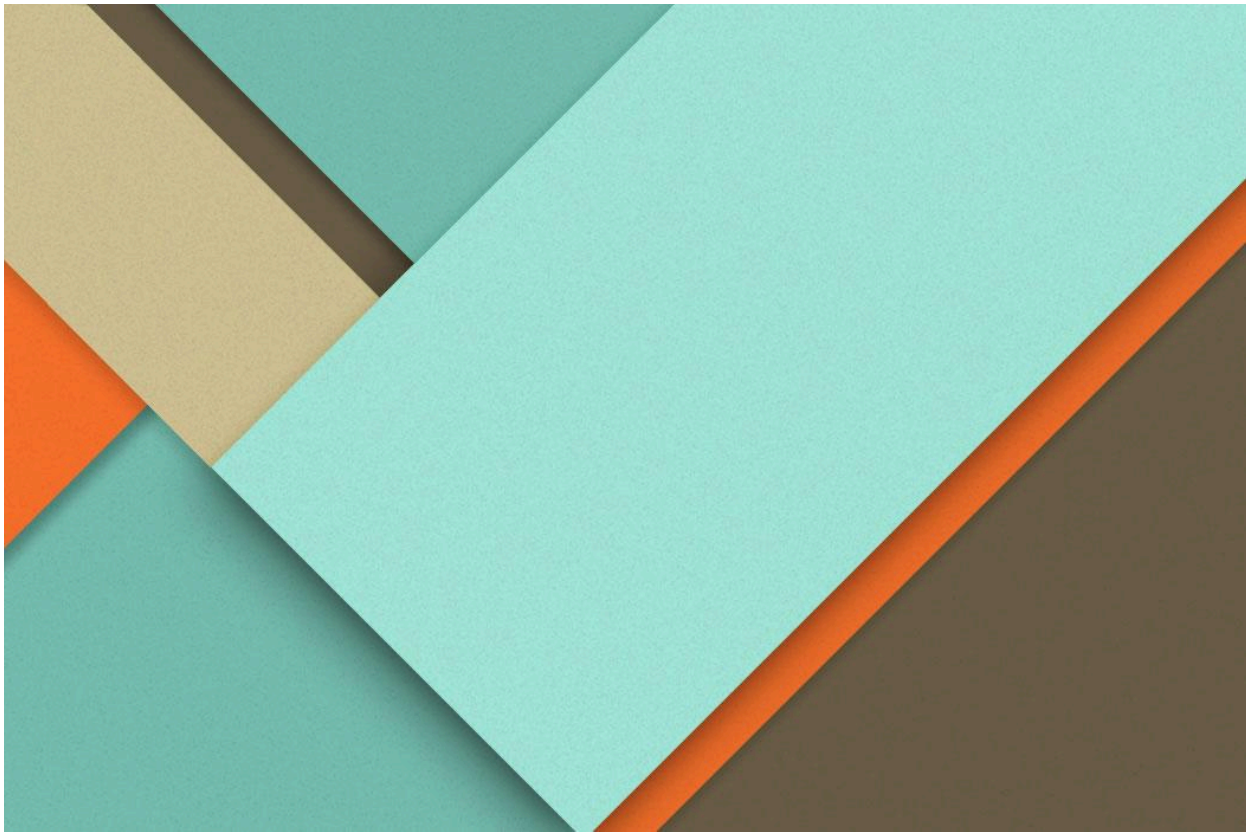


CSTP2301 Final Project V3

16.03.2024

ML

Jia Xi Lin

Muochu Hu

Goals

1. Picking IDs 542236, 67321, 549295, 41108, 54982 to do the calculations.
2. Using a regression model to train and test the data set.
3. Comparing the actual value of y and predicted value of y and calculate the accuracy of them, fill the results in an Excel file.

Step 1: Read the Dataset

Begin by importing necessary libraries and loading the dataset. Use pandas to read the CSV file containing the data. This step is crucial as it sets up your data for preprocessing and analysis.

```
import pandas as pd
data = pd.read_csv('Sub_Oil_VLCC_Monthly.csv')
data
```

Step 2: Data Preprocessing

Prepare the dataset for the model. This involves splitting the dataset into features (X) and target variables (y). Since the model requires numerical inputs, ensure that the data is cleaned and appropriately formatted.

- Exclude the last row from the dataset for features (X).
- Define target columns for various scenarios (good, mid, and bad results).
- Exclude the first row from target columns to align with X.

To generate X and y, we can consider the entire table as X and copy the target column in a vector as y. Remember that the label of sample t in X is in row t+1 in y.

```
#Splitting the data into training and testing data

X = data[:-1]  #whole data exclude the last row as X

#target columns
good_result1_target_column = data['542236']
good_result2_target_column = data['67321']
mid_result_target_column = data['549295']
bad_result1_target_column = data['41108']
bad_result2_target_column = data['541982']

#target column exclude the first row as y
good1_y = good_result1_target_column[1:]
good2_y = good_result2_target_column[1:]
mid_y = mid_result_target_column[1:]
bad1_y = bad_result1_target_column[1:]
bad2_y = bad_result2_target_column[1:]
```

Other than that, we apply normalization to X.

```
#Normalization
#get the first column of the data
time = data.iloc[:-1, 0].astype(int)

#Normalize the data
scaledX = data.iloc[:-1, 1:]  # Excluding the last row and first column for dates
scaledX = (scaledX - scaledX.min()) / (scaledX.max() - scaledX.min())

# Assign the 'date' column back as integers
scaledX.insert(0,'Unnamed: 0' ,time)

scaledX_train = scaledX.iloc[:n, :] #scaled training data
scaledX_test = scaledX.iloc[n:, :] #scaled testing data
```

Step 3: Splitting the Data into Training and Testing Sets

It's essential to evaluate the model's performance on unseen data. Split the dataset into a training set used for learning and a testing set for evaluation. We aim to assess the model's accuracy over the last three years available in the dataset. To achieve this, consider training the model on all samples from the beginning up to n-36, and then test it on the last 36 samples (n is the total number of samples excluding the last one that doesn’t include a label)

```python
n = len(X)-36 #number of data points for training and testing

X_train = X.iloc[:n, :] #training data
X_test = X.iloc[n:, :] #testing data

#good1 target column training and testing data
good1_y_train = good1_y.iloc[:n]
good1_y_test = good1_y.iloc[n:]

#good2 target column training and testing data
good2_y_train = good2_y.iloc[:n]
good2_y_test = good2_y.iloc[n:]

#mid target column training and testing data
mid_y_train = mid_y.iloc[:n]
mid_y_test = mid_y.iloc[n:]

#bad1 target column training and testing data
bad1_y_train = bad1_y.iloc[:n]
bad1_y_test = bad1_y.iloc[n:]

#bad2 target column training and testing data
bad2_y_train = bad2_y.iloc[:n]
bad2_y_test = bad2_y.iloc[n:]
```

Making sure of the training data are the data from 19910101 to 20201201, and the testing data are from 20210101 to 20231201 by printing them out.

```python
print("X_train_Head: ", X_train.iloc[0,0])
print("X_train_Tail: ", X_train.iloc[-1,0])
print("X_test_Head: ", X_test.iloc[0,0])
print("X_test_Tail: ", X_test.iloc[-1,0])
print("scaledX_train_Head: ", scaledX_train.iloc[0,0])
print("scaledX_train_Tail: ", scaledX_train.iloc[-1,0])
print("scaledX_test_Head: ", scaledX_test.iloc[0,0])
print("scaledX_test_Tail: ", scaledX_test.iloc[-1,0])
```

[22]

```
... X_train_Head:  19910101
    X_train_Tail:  20201201
    X_test_Head:  20210101
    X_test_Tail:  20231201
    scaledX_train_Head:  19910101
    scaledX_train_Tail:  20201201
    scaledX_test_Head:  20210101
    scaledX_test_Tail:  20231201
```

Step 4: Model Selection and Training

We designed a model_fit_predict function to simplify the process of training and predicting with various regression models. This function receives the training data and a target variable, then applies several algorithms to produce a dictionary of predictions.

We tested five models: Decision Tree Regressor, Linear Regression, Lasso Regression, Support Vector Regression, and Multi-Layer Perceptron Regressor, plus a Gradient Boosting Regressor.After fitting each model to the training data, we invoked the predict method using the test set (X_test) to generate predictions.

By organizing predictions in a dictionary with model names as keys, we facilitated the subsequent evaluation of model performance based on accuracy metrics. This structured approach was instrumental in identifying the most suitable model for each target column, leading us to a set of predictions that could then be analyzed for accuracy and reliability.

```python
# The models we use to do the prediction
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import GradientBoostingRegressor

def model_fit_predict(y_train):
    predictions = {}  # Initialize an empty dictionary

    # Decision Tree Regressor
    DT = DecisionTreeRegressor(random_state=42)
    DT.fit(X_train, y_train)
    predictions['DT'] = DT.predict(X_test)

    # Linear Regression
    LR = LinearRegression()
    LR.fit(X_train, y_train)
    predictions['LR'] = LR.predict(X_test)

    # Lasso Regression
    Lasso_model = Lasso()
    Lasso_model.fit(X_train, y_train)
    predictions['Lasso'] = Lasso_model.predict(X_test)

    # Support Vector Regression
    SVR_model = SVR()
    SVR_model.fit(X_train, y_train)
    predictions['SVR'] = SVR_model.predict(X_test)

    # Multi-Layer Perceptron Regressor
    MLP = MLPRegressor(hidden_layer_sizes=(350,150,50), max_iter=1000)
    MLP.fit(X_train, y_train)
    predictions['MLP'] = MLP.predict(X_test)

    # Gradient Boosting Regressor
    GBR = GradientBoostingRegressor()
    GBR.fit(X_train, y_train)
    predictions['GBR'] = GBR.predict(X_test)

    return predictions

good1_predictions = model_fit_predict(good1_y_train)
good2_predictions = model_fit_predict(good2_y_train)
mid_predictions = model_fit_predict(mid_y_train)
bad1_predictions = model_fit_predict(bad1_y_train)
bad2_predictions = model_fit_predict(bad2_y_train)
```

And this time we have set some hyper parameters for the MLP model, we have set 3 layers, the first layer has 350 neurons, second layer has 150 neurons and the third layer has 50 neurons and set the maximum number of iterations for 1000 times.

```
# Random Forest Regressor
RFR = RandomForestRegressor()
RFR.fit(X_train, y_train)
predictions['RFR'] = RFR.predict(X_test)
predictions['Scaled RFR'] = RFR.predict(scaledX_test)

#Ridge Regression
RR = Ridge()
RR.fit(X_train, y_train)
predictions['RR'] = RR.predict(X_test)
predictions['Scaled RR'] = RR.predict(scaledX_test)
```

Also, we add two more models, Random Forest Regression and Ridge Regression.

```
{'DT': array([26190.52, 26190.52, 25924.44, 26190.52, 23370.15, 26190.52,
       26190.52, 27437.5 , 27437.5 , 25073.01, 28510.48, 27437.5 ,
       27437.5 , 27437.5 , 23370.15, 25073.01, 27437.5 , 28510.48,
       28510.48, 33000.  , 40000.  , 44355.77, 64904.01, 40159.3 ,
       50875.  , 60000.  , 64904.01, 40159.3 , 40000.  , 40000.  ,
       41356.63, 40159.3 , 64904.01, 40159.3 , 64904.01, 64904.01]), 'Scaled DT': array([22199.43, 22199.43, 22199.43, 22199.43, 22199.43, 2219
       22199.43, 22199.43, 22199.43, 22199.43, 22199.43, 22199.43,
       22199.43, 22199.43, 22199.43, 22199.43, 22199.43, 22199.43,
       22199.43, 22199.43, 22199.43, 22199.43, 22199.43, 22199.43,
       22199.43, 22199.43, 22199.43, 22199.43, 22199.43, 22199.43,
       22199.43, 22199.43, 22199.43, 22199.43, 22199.43, 22199.43]), 'LR': array([  47639.62406022,   76291.28892127,   84007.60211534,
         84367.32291944,   92340.22236385,   54781.46611355,
         61810.77499469,   45423.7240802 ,   61840.48634965,
         97622.29592426,   68620.39617335,   -5656.78853467,
         12197.3138489 ,   22208.96454527,   48551.26685627,
         83029.64500493,   25410.53970033,  -78984.27977377,
       -192600.59306291,  -74258.57026851,    4273.77436649,
         14767.37497133,   16667.95001831,  -42587.65764038,
       -279896.2774728 , -262251.89454046, -199466.66852274,
       -175483.77846867, -113652.21636635, -163821.03185074,
       -122377.8581551 , -104954.96245413, -112133.43853761,
       -216143.46976441, -282867.95830059, -185646.65056343]), 'Scaled LR': array([1247071.58234314, 1244805.69151451, 1246037.83309938,
       1247818.21169995, 1246960.90729249, 1246487.20838292,
       1246686.56325345, 1246344.46197804, 1245019.89119879,
       1245875.49371877, 1246433.72159861, 1246596.35367094,
...
       5849.19942254, 5728.95551055, 5607.29745844, 5658.80839698,
       2656.23933432, 2757.04597246, 2611.61841449, 2586.49487566,
       2599.81927098, 2596.35187308, 2769.74445544, 2866.69860161,
       2654.85138877, 2509.22646811, 2555.92105843, 2476.19071751])}
```

Step 5: Model Prediction and Evaluation

The calculate_accuracies function measures how well our models predict outcomes. It measures the difference between what the model predicts and the actual results, turning this into a percentage that represents the model's accuracy. This is done for each prediction and then averaged out to get the model's overall accuracy. We've applied this function to several datasets, allowing us to see which model gives the most accurate predictions for each case. The results are kept neat, with individual prediction accuracies and the model's average accuracy neatly recorded and easy to compare.

```
import numpy as np
def calculate_accuracies(predictions, y_test):
    average_accuracies = {}
    individual_accuracies = {}

    for model, model_predictions in predictions.items():
        # Calculate accuracy for each prediction
        accuracies = (1 - np.abs(model_predictions - y_test) / y_test) * 100

        # Calculate average accuracy for the model
        average_accuracy = np.mean(accuracies)

        # Store the average accuracy, rounded to two decimal places
        average_accuracies[model] = round(average_accuracy, 2)

        # Store individual accuracies
        individual_accuracies[model] = accuracies

    return average_accuracies, individual_accuracies

# Calculate accuracies
good1_avg_acc, good1_ind_acc = calculate_accuracies(good1_predictions, good1_y_test)
good2_avg_acc, good2_ind_acc = calculate_accuracies(good2_predictions, good2_y_test)
mid_avg_acc, mid_ind_acc = calculate_accuracies(mid_predictions, mid_y_test)
bad1_avg_acc, bad1_ind_acc = calculate_accuracies(bad1_predictions, bad1_y_test)
bad2_avg_acc, bad2_ind_acc = calculate_accuracies(bad2_predictions, bad2_y_test)
```

After the accuracies are calculated, the next step is to find the best accuracy for each model.

```
#pick the best model for each target column
good1_best_model = max(good1_avg_acc, key=good1_avg_acc.get)
good2_best_model = max(good2_avg_acc, key=good2_avg_acc.get)
mid_best_model = max(mid_avg_acc, key=mid_avg_acc.get)
bad1_best_model = max(bad1_avg_acc, key=bad1_avg_acc.get)
bad2_best_model = max(bad2_avg_acc, key=bad2_avg_acc.get)
```

We can easily find out the best model for each target ID by printing the results:

Before fill in hyper parameters for MLP models:

```
print("Avg_Good1: ", good1_avg_acc)
#print(good2_ind_acc)
print("Avg_Good2: ", good2_avg_acc)
#print(mid_ind_acc)
print("Avg_Mid: ", mid_avg_acc)
#print(bad1_ind_acc)
print("Avg_Bad1: ", bad1_avg_acc)
#print(bad2_ind_acc)
print("Avg_Bad2: ", bad2_avg_acc)

print("Best_Good1: ", good1_best_model, good1_avg_acc[good1_best_model])
print("Best_Good2: ", good2_best_model, good2_avg_acc[good2_best_model])
print("Best_Mid: ", mid_best_model, mid_avg_acc[mid_best_model])
print("Best_Bad1: ", bad1_best_model, bad1_avg_acc[bad1_best_model])
print("Best_Bad2: ", bad2_best_model, bad2_avg_acc[bad2_best_model])
✓ 0.0s
```

```
Avg_Good1:  {'DT': 84.79, 'LR': -183.68, 'Lasso': 17.35, 'SVR': 61.08, 'MLP': -313.58, 'GBR': 86.2}
Avg_Good2:  {'DT': 96.47, 'LR': 70.49, 'Lasso': 94.69, 'SVR': 79.18, 'MLP': -130877.2, 'GBR': 97.81}
Avg_Mid:  {'DT': 79.45, 'LR': -212.78, 'Lasso': 78.32, 'SVR': 69.69, 'MLP': -707486.15, 'GBR': 87.05}
Avg_Bad1:  {'DT': 44.27, 'LR': -734.57, 'Lasso': 63.81, 'SVR': 70.8, 'MLP': -203581.89, 'GBR': 65.88}
Avg_Bad2:  {'DT': 27.04, 'LR': -1504.31, 'Lasso': 8.65, 'SVR': 61.08, 'MLP': -565461.58, 'GBR': 55.55}
Best_Good1:  GBR 86.2
Best_Good2:  GBR 97.81
Best_Mid:  GBR 87.05
Best_Bad1:  SVR 70.8
Best_Bad2:  SVR 61.08
```

After apply normalization, fill in hyper parameters for MLP models and add Random Forest Regression and Ridge Regression.:

```
Avg_Good1:  {'DT': 84.79, 'Scaled DT': 70.26, 'LR': -183.68, 'Scaled LR': -3750.15, 'Lasso': 17.35, 'Scaled Lasso': -128.88, 'SVR': 61.08, '
Avg_Good2:  {'DT': 96.47, 'Scaled DT': 58.97, 'LR': 70.49, 'Scaled LR': -220.21, 'Lasso': 94.69, 'Scaled Lasso': 4.24, 'SVR': 79.18, 'Scaled
Avg_Mid:  {'DT': 79.45, 'Scaled DT': 63.59, 'LR': -212.78, 'Scaled LR': -4088.68, 'Lasso': 78.32, 'Scaled Lasso': -116.99, 'SVR': 69.69, 'Sc
Avg_Bad1:  {'DT': 44.27, 'Scaled DT': 70.41, 'LR': -734.57, 'Scaled LR': -10228.91, 'Lasso': 63.81, 'Scaled Lasso': 1.12, 'SVR': 70.8, 'Scal
Avg_Bad2:  {'DT': 27.04, 'Scaled DT': 63.68, 'LR': -1504.31, 'Scaled LR': -13450.75, 'Lasso': 8.65, 'Scaled Lasso': -610.52, 'SVR': 61.08, '
Best_Good1:  GBR 89.92
Best_Good2:  GBR 97.5
Best_Mid:  RFR 88.42
Best_Bad1:  Scaled RFR 75.1
Best_Bad2:  Scaled RFR 72.71
```

We can find out that after we set some layers for the MLP models the accuracy is improved but still negative, so we stop trying to change the value of the hyper parameters, we think MLP is not really fit for this case.

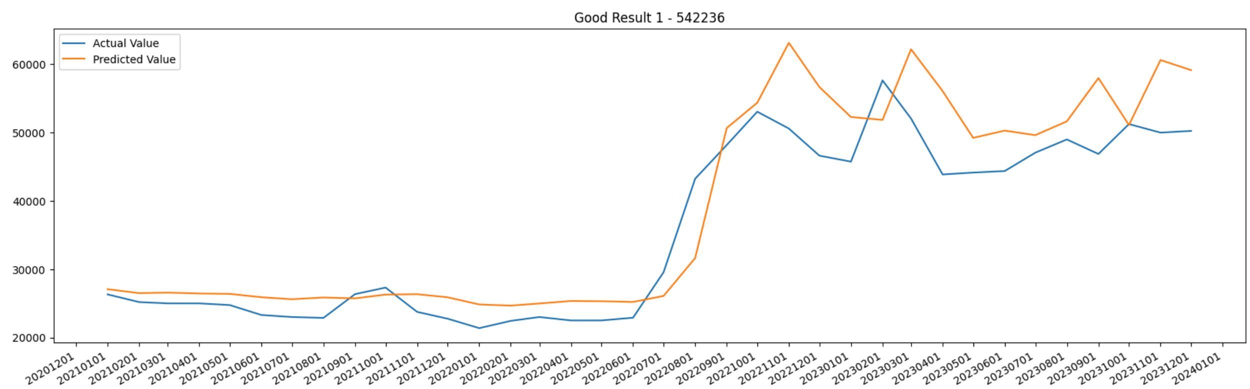
Summary:

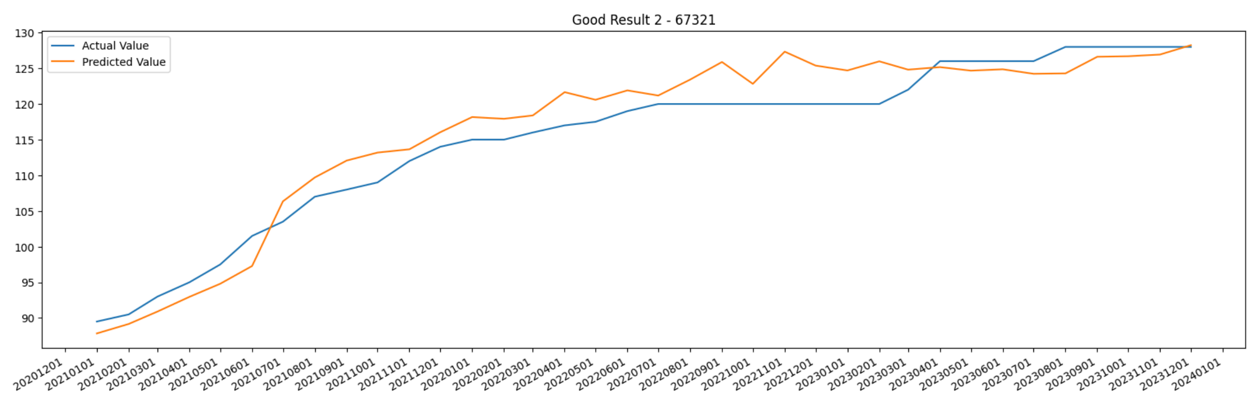
ID	Best Model	Accuracy
542236	Gradient Boosting Regression	88.16%
67321	Gradient Boosting Regression	97.5%
549295	Random Forest Regression	88.86%
41108	Random Forest Regression + Normalization	75.7%
541982	Random Forest Regression + Normalization	70.52%

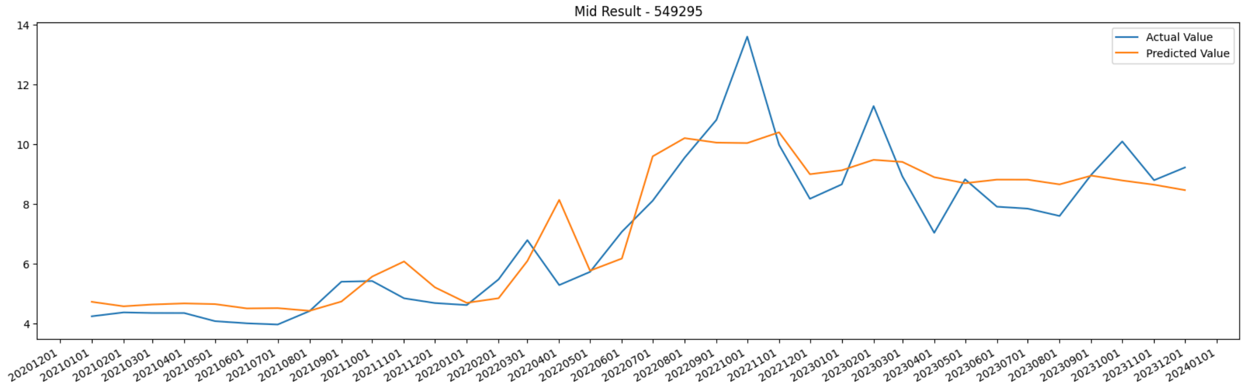
Step 6: Plot Diagrams

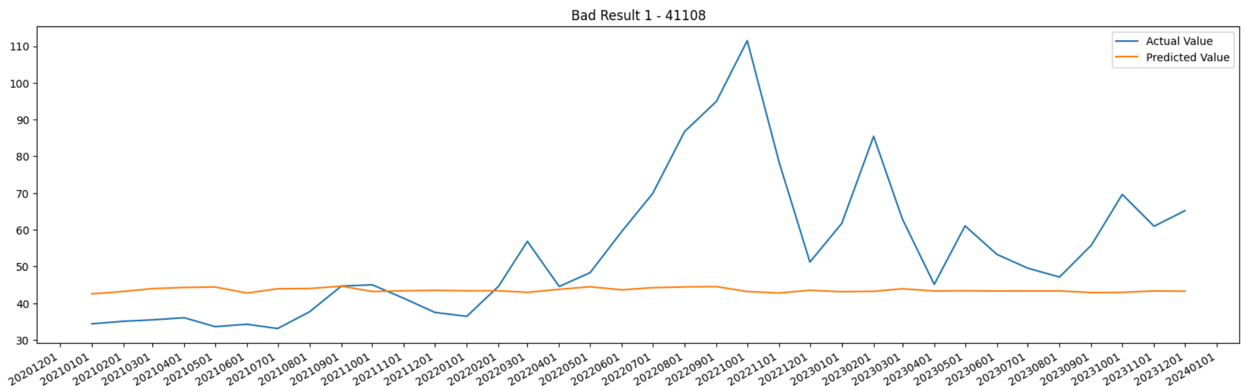
Using the Matplotlib library for plotting the performance of prediction models over time. The script defines a function called plot_best_model.These plots are critical for visually assessing how well each model's predictions align with the actual data, which is an essential part of the model evaluation process.

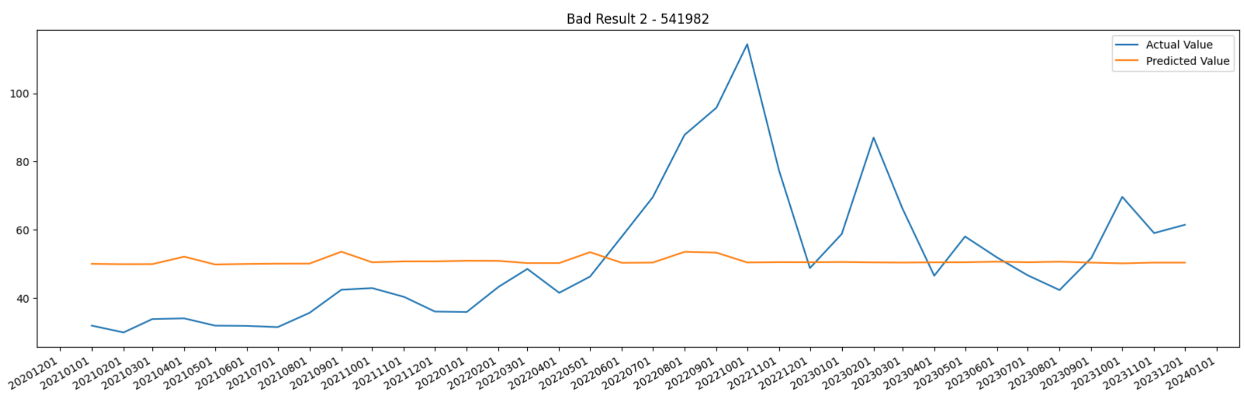
```python
# plot a graph of the best model for each target column that show the prediction and the actual value
#x axis is the index of the testing data
#y axis is the actual value and the predicted value
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from datetime import datetime

def plot_best_model(best_model, y_test, predictions, title):
    X_test['Datetime'] = pd.to_datetime(X_test['Unnamed: 0'], format='%Y%m%d')
    plt.figure(figsize=(20,6))
    plt.plot(X_test['Datetime'], y_test, label='Actual Value')
    plt.plot(X_test['Datetime'], predictions[best_model], label='Predicted Value')
    plt.title(title)
    plt.legend()
     # Format the dates on the x-axis
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y%m%d'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator())

    # Rotate and align the tick labels so they look better
    plt.gcf().autofmt_xdate()
    plt.show()

plot_best_model(good1_best_model, good1_y_test, good1_predictions, 'Good Result 1 - 542236')
plot_best_model(good2_best_model, good2_y_test, good2_predictions, 'Good Result 2 - 67321')
plot_best_model(mid_best_model, mid_y_test, mid_predictions, 'Mid Result - 549295')
plot_best_model(bad1_best_model, bad1_y_test, bad1_predictions, 'Bad Result 1 - 41108')
plot_best_model(bad2_best_model, bad2_y_test, bad2_predictions, 'Bad Result 2 - 541982')
```











Step 7: Export csv files

we devised the write_csv function. This function takes in the actual values, the predicted values, the individual prediction accuracies, the average accuracy, and the desired filename for output.

For every model, we used the function to write a CSV file containing the time series from our test set, along with the corresponding actual and predicted values, and their accuracies. It neatly places the average accuracy at the top of the column, avoiding repetition, for a cleaner look in the data file.

```python
# write an csv file
# first colume is the time series, this is first column of the X_test
# second column is the actual value, this is the y_test
# third column is the predicted value, this is the y_pred
# fourth column is the accuracy, this is the accuracy
# fifth column is the average accuracy, this is the avg_accuracy

import csv

def write_csv(y_test, y_pred, ind_acc, avg_acc, filename):
    with open(filename, 'w', newline='') as file:
        X_test_local = X_test.set_index('Unnamed: 0')
        writer = csv.writer(file)
        writer.writerow(["Time Series", "Target Value", "Predicted Value", "Accuracy", "Average Accuracy"])
        for i in range(len(y_test)):
            if i == 0:
                writer.writerow([X_test_local.index[i], y_test.iloc[i], y_pred[i], ind_acc.loc[361+i], avg_acc])
            else:
                writer.writerow([X_test_local.index[i], y_test.iloc[i], y_pred[i], ind_acc.loc[361+i], ""])

write_csv(good1_y_test, good1_predictions[good1_best_model], good1_ind_acc[good1_best_model], good1_avg_acc[good1_best_model], '542236_good1.csv')
write_csv(good2_y_test, good2_predictions[good2_best_model], good2_ind_acc[good2_best_model], good2_avg_acc[good2_best_model], '67321_good2.csv')
write_csv(mid_y_test, mid_predictions[mid_best_model], mid_ind_acc[mid_best_model], mid_avg_acc[mid_best_model], '549295_mid.csv')
write_csv(bad1_y_test, bad1_predictions[bad1_best_model], bad1_ind_acc[bad1_best_model], bad1_avg_acc[bad1_best_model], '41108_bad1.csv')
write_csv(bad2_y_test, bad2_predictions[bad2_best_model], bad2_ind_acc[bad2_best_model], bad2_avg_acc[bad2_best_model], '541982_bad2.csv')
```