**Data set 3 : Best model:** Enhanced Stacking ML Models.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **RMSE** | **MAE** | **MAD** | **MAPE** | **Best Params** | **Tuning Method** |
| Linear regression | 2.13 | 1.50 | 1.50 | 2.39 | intercept:0.4658697729251742  Estiamted coefficients:[ 6.94174189e-01 1.79724508e-01 4.31945182e-02 3.98989702e-02  1.89830751e-02 -5.36173487e-03 -1.21644652e-02 2.70335052e-03  3.62245994e-02 -5.65311557e-02 2.62056403e-02 5.62515541e-02  7.29996152e-02 -4.49928579e-02 -2.28521922e-03 -1.20967388e-02  -2.77147705e-02 -1.36407494e-02 -1.12213412e-02 2.16036301e-02  -3.50800532e-02 7.24726603e-02 -3.61021478e-02 -1.05215586e-04  -2.67268402e-02 -2.58123844e-03 3.05725813e-02 -1.77690149e-02  1.19199187e-01 -1.16122144e-01] |  |
| Linear regression-Optimized | 2.13 | 1.50 | 1.50 | 2.39 | Best Parameters: {'fit\_intercept': True}  Coefficients: [ 6.94174189e-01 1.79724508e-01 4.31945182e-02 3.98989702e-02  1.89830751e-02 -5.36173487e-03 -1.21644652e-02 2.70335052e-03  3.62245994e-02 -5.65311557e-02 2.62056403e-02 5.62515541e-02  7.29996152e-02 -4.49928579e-02 -2.28521922e-03 -1.20967388e-02  -2.77147705e-02 -1.36407494e-02 -1.12213412e-02 2.16036301e-02  -3.50800532e-02 7.24726603e-02 -3.61021478e-02 -1.05215586e-04  -2.67268402e-02 -2.58123844e-03 3.05725813e-02 -1.77690149e-02  1.19199187e-01 -1.16122144e-01]  Intercept: 0.4658697729251742 | Random Search 'fit\_intercept': [True, False] |
| Polynomial Regression -PCA | 5.63 | 4.24 | 3.09 | 7.81% | Best Polynomial degree: 3 |  |
| Polynomial Regression-Optimized- PCA | 2.19 | 1.57 | 1.13 | 2.42% | {'pca\_\_n\_components': 12, 'poly\_\_degree': 1} | Random Search 'poly\_\_degree': randint(1, 6) |
| Gaussian process | 7.768 | 3.373 | 1.498 | 6.006% | kernel = RBF(length\_scale=1.0) |  |
| Gaussian process-Optimized- | 6.35 | 2.62 | 1.35 | 7.33 | {'kernel\_\_k1\_\_constant\_value': 4.419450186421157, 'kernel\_\_k2\_\_length\_scale': 3.0122914019804194} | Random Search  param\_dist = {      'kernel\_\_k1\_\_constant\_value': uniform(0.1, 10.0),      'kernel\_\_k2\_\_length\_scale': uniform(0.1, 10.0),  } |
| MA-Uni | 19.31 | 15.88 | 15.88 | 29.90% | window\_size = 5 |  |
| MA-Uni-Optimized | 15.081 | 12.204 | 12.204 | 21.268% | Best Window Size: 2 | Random Search  window\_sizes = range(2, 21) |
| MA-Multi | 19.20 | 15.78 | 15.78 | 29.66% | window\_size = 5 |  |
| MA-Multi-Optimized | 15.32 | 12.35 | 12.35 | 21.92% | Best Window Size: 2 | Random Search  window\_sizes = range(2, 21) |
| Arima-Uni | 25.40 | 20.04 | 20.04 | 44.41 | p, d, q = 5, 1, 0 |  |
| Arima-Uni-Optimized | 21.460 | 17.225 | 17.225 | 34.228% | Best ARIMA(2, 1, 3) | Random Search  p\_range = range(0, 6)  d\_range = range(0, 3)  q\_range = range(0, 6) |
| Arima-Multi | 6.48 | 5.56 | 5.56 | 9.69% | p, d, q = 4, 2, 5 |  |
| Arima-Multi-Optimized | 2.122 | 1.493 | 1.493 | 2.328% | Best SARIMAX(5, 1, 5) | Random Search  p\_range = range(0, 6)  d\_range = range(0, 3)  q\_range = range(0, 6) |
| Stacking- With best STmodels | 2.144 | 1.55 | 1.18 | R-squared: 0.99 | With best parameters of   ('gp', gp\_model),          ('poly', poly\_model),          ('sarimax', sarimax\_model),          ('lr', lr\_model) | # Create Gaussian Process Model  kernel = C(4.41, (1e-2, 1e2)) \* RBF(3.01, (1e-2, 1e2))  gp\_model = GaussianProcessRegressor(kernel=kernel, n\_restarts\_optimizer=9)  # Initialize models  sarimax\_model = CustomSARIMAX(order=(5, 1, 5))  poly\_model = CustomPolynomialRegression(degree=2)  lr\_model = LinearRegression() |
| Stacking- With best ST models-PCA | RMSE: 2.141 | MAE: 1.55 | MAD: 1.16 | R-squared: 0.99 | With best parameters of   ('gp', gp\_model),          ('poly', poly\_model),          ('sarimax', sarimax\_model),          ('lr', lr\_model) | # Create Gaussian Process Model  kernel = C(4.41, (1e-2, 1e2)) \* RBF(3.01, (1e-2, 1e2))  gp\_model = GaussianProcessRegressor(kernel=kernel, n\_restarts\_optimizer=9)  # Initialize models  sarimax\_model = CustomSARIMAX(order=(5, 1, 5))  poly\_model = CustomPolynomialRegression(degree=2)  lr\_model = LinearRegression() |
| Stacking with random forest | RMSE: 2.36 | MAE: 1.70 | MAD: 1.27 | R-squared: 0.98 | estimators=[          ('gp', gp\_model),          ('poly', poly\_model),          ('sarimax', sarimax\_model),          ('lr', lr\_model)      ],      final\_estimator=RandomForestRegressor(n\_estimators=100, random\_state=42) | # Gaussian Process Model  kernel = C(4.41, (1e-2, 1e2)) \* RBF(3.01, (1e-2, 1e2))  gp\_model = GaussianProcessRegressor(kernel=kernel, n\_restarts\_optimizer=9)  # Initialize models  sarimax\_model = CustomSARIMAX(order=(5, 1, 5))  poly\_model = CustomPCA\_PolynomialRegression(degree=2, n\_components=10)  lr\_model = LinearRegression()  # Stacking Ensemble with a more complex final estimator  stacked\_model = StackingRegressor(      estimators=[          ('gp', gp\_model),          ('poly', poly\_model),          ('sarimax', sarimax\_model),          ('lr', lr\_model)      ],      final\_estimator=RandomForestRegressor(n\_estimators=100, random\_state=42)  ) |
| Stacking with AdaBoost | RMSE: 2.44 | MAE: 1.81 | MAD: 1.39 | R-squared: 0.98 | estimators=[          ('gp', gp\_model),  # Gaussian Process          ('poly', poly\_model),  # Polynomial Regression with PCA          ('sarimax', sarimax\_model),  # Custom SARIMAX          ('lr', lr\_model)  # Linear Regression      ],      final\_estimator=AdaBoostRegressor(          base\_estimator=DecisionTreeRegressor(max\_depth=4),          n\_estimators=100,          random\_state=42,          learning\_rate=0.05 | stacked\_model = StackingRegressor(      estimators=[          ('gp', gp\_model),  # Gaussian Process          ('poly', poly\_model),  # Polynomial Regression with PCA          ('sarimax', sarimax\_model),  # Custom SARIMAX          ('lr', lr\_model)  # Linear Regression      ],      final\_estimator=AdaBoostRegressor(          base\_estimator=DecisionTreeRegressor(max\_depth=4),          n\_estimators=100,          random\_state=42,          learning\_rate=0.05      )  ) |
| KNN | 2.67 | 1.73 | 1.14 | 3.41 | Number of Neighbors: 5 | n\_neighbors = [3, 5, 7, 9, 11, 13, 15, 17] |
| KNN-Optimized | 2.72 | 1.88 | 1.22 | None% | {'n\_neighbors': 10, 'p': 2, 'weights': 'distance'} | Random Search  param\_distributions = {      'n\_neighbors': randint(1, 31),      'weights': ['uniform', 'distance'],      'p': [1, 2]  } |
| Decision Tree | 2.29 | 1.69 | 1.27 | 2.81 | Max Depth: 5 | max\_depths = [3, 5, 7, 9, 11, 13, 15, 17] |
| Decision Tree-Optimized | 2.40 | 1.72 | 1.31 | 3.00% | {'max\_depth': 5, 'min\_samples\_leaf': 5, 'min\_samples\_split': 8} | Random Search  param\_dist = {      'max\_depth': randint(3, 20),      'min\_samples\_split': randint(2, 20),      'min\_samples\_leaf': randint(1, 20)  } |
| SVR | 2.16 | 1.63 | 1.27 | 2.87% | (kernel='rbf', C=1.0, epsilon=0.1) | kernel='rbf', C=1.0, epsilon=0.1 |
| SVR-Optimized | 1.97 | 1.40 | 0.98 | 2.35% | {'C': 3.4167643418329705, 'epsilon': 0.04115113049561093, 'gamma': 0.025764174425233172} | Random Search  param\_dist = {      'C': loguniform(1e-2, 1e2),      'gamma': loguniform(1e-4, 1e-1),      'epsilon': loguniform(1e-4, 1e-1)  } |
| Stacking-Best ML models | RMSE: 2.0999 | MAE: 1.5532 | MAD: 1.2165 | MAPE: -2.5405% | With best parameters of    ('knn', knn),  ('decision\_tree', decision\_tree),  ('svr', svr) | # Decision Tree with optimized parameters  decision\_tree = DecisionTreeRegressor(max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=8)  # KNN with optimized parameters  knn = KNeighborsRegressor(n\_neighbors=10, p=2, weights='distance')  # SVR with optimized parameters  svr = make\_pipeline(StandardScaler(), SVR(C=3.4167643418329705, epsilon=0.04115113049561093, gamma=0.025764174425233172)) |
| Ensemble with Random Forrest | Random Forest Model Results:  RMSE: 2.1543, | MAE: 1.4572 | MAD: 1.0075 |  | ('knn', knn),  ('decision\_tree', decision\_tree),  ('svr', svr) | # Decision Tree with optimized parameters  decision\_tree = DecisionTreeRegressor(max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=8)  # KNN with optimized parameters  knn = KNeighborsRegressor(n\_neighbors=10, p=2, weights='distance')  # SVR with optimized parameters and scaling  svr = make\_pipeline(StandardScaler(), SVR(C=3.4167643418329705, epsilon=0.04115113049561093, gamma=0.025764174425233172))  # Random Forest model  random\_forest = RandomForestRegressor(n\_estimators=100, random\_state=42)  # Stacking Ensemble with a different final estimator  stacked\_model = StackingRegressor(      estimators=[          ('knn', knn),          ('decision\_tree', decision\_tree),          ('svr', svr),          ('random\_forest', random\_forest)      ],      final\_estimator=Ridge(alpha=1.0)  ) |
| Stacking-Best ML models-PCA | RMSE: 2.4738 | MAE: 1.7622 | MAD: 1.3588 | MAPE: -2.9802% | With best parameters of   ('knn', knn),  ('decision\_tree', decision\_tree),  ('svr', svr) | # Decision Tree with optimized parameters  decision\_tree = DecisionTreeRegressor(max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=8)  # KNN with optimized parameters  knn = KNeighborsRegressor(n\_neighbors=10, p=2, weights='distance')  # SVR with optimized parameters  svr = make\_pipeline(StandardScaler(), SVR(C=3.4167643418329705, epsilon=0.04115113049561093, gamma=0.025764174425233172)) |
| Enhanced Stacking Model | RMSE: 1.93 | Test MAE: 1.40 | Test MAD: 1.06 |  | With best parameters of   ('knn', knn),  ('decision\_tree', decision\_tree),  ('svr', svr) and adding random forest and gradient boosting  decision\_tree = DecisionTreeRegressor(max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=8)  knn = KNeighborsRegressor(n\_neighbors=10, p=2, weights='distance')  svr = make\_pipeline(StandardScaler(), SVR(C=3.4167643418329705, epsilon=0.04115113049561093, gamma=0.025764174425233172))  random\_forest = RandomForestRegressor(n\_estimators=100, random\_state=42)  gradient\_boosting = GradientBoostingRegressor(n\_estimators=100) | # Initialize base models with optimized parameters  decision\_tree = DecisionTreeRegressor(max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=8)  knn = KNeighborsRegressor(n\_neighbors=10, p=2, weights='distance')  svr = make\_pipeline(StandardScaler(), SVR(C=3.4167643418329705, epsilon=0.04115113049561093, gamma=0.025764174425233172))  random\_forest = RandomForestRegressor(n\_estimators=100, random\_state=42)  gradient\_boosting = GradientBoostingRegressor(n\_estimators=100) |
| GRU | 22.0018 | 17.9781 | 17.8068 | 33.6883% | Lag : 24  Dropout(rate=0.1)  First GRU layer: 200 units  Second GRU layer: 100 units  Third GRU layer: 50 units  Fourth GRU layer: 50 units |  |
| GRU-Optimized | 21.8340 | 17.7258 | 17.7137 | 33.9167% | Lag : 24,{'model\_\_gru\_units4': 50, 'model\_\_gru\_units3': 25, 'model\_\_gru\_units2': 150, 'model\_\_gru\_units1': 200, 'model\_\_dropout\_rate': 0.5} | Random Search      'model\_\_gru\_units1': [100, 200, 300],      'model\_\_gru\_units2': [50, 100, 150],      'model\_\_gru\_units3': [25, 50, 75],      'model\_\_gru\_units4': [25, 50],      'model\_\_dropout\_rate': [0.1, 0.2, 0.3, 0.4, 0.5] |
| LSTM | 21.8089 | 17.6512 | 17.6523 | 34.1376% | Lag : 24  Dropout(rate=0.1)  First LSTM layer: 200 units  Second LSTM layer: 100 units  Third LSTM layer: 50 units  Fourth (last) LSTM layer: 50 units |  |
| LSTM-Optimized | 21.8088 | 17.6202 | 17.6201 | 34.2585% | Lag : 24,{'model\_\_lstm\_units4': 50, 'model\_\_lstm\_units3': 25, 'model\_\_lstm\_units2': 100, 'model\_\_lstm\_units1': 200, 'model\_\_dropout\_rate': 0.1} | Random Search      'model\_\_lstm\_units1': [100, 200],      'model\_\_lstm\_units2': [50, 100],      'model\_\_lstm\_units3': [25, 50, 75],      'model\_\_lstm\_units4': [25, 50],      'model\_\_dropout\_rate': [0.1, 0.2] |
| BILSTM | 22.0813 | 17.4903 | 17.4890 | 35.7562% | Lag : 24  Dropout(rate=0.1)  First BILSTM layer: 200 units  Second BILSTM layer: 100 units  Third BILSTM layer: 50 units  Fourth (last) BILSTM layer: 50 units |  |
| BILSTM-Optimized | 21.8414 | 17.7729 | 17.7447 | 33.8098% | Lag : 24,{'model\_\_dropout\_rate': 0.1, 'model\_\_bilstm\_units4': 25, 'model\_\_bilstm\_units3': 25, 'model\_\_bilstm\_units2': 100, 'model\_\_bilstm\_units1': 100} | Random Search      'model\_\_bilstm\_units1': [100, 200],      'model\_\_bilstm\_units2': [50, 100],      'model\_\_bilstm\_units3': [25, 50, 75],      'model\_\_bilstm\_units4': [25, 50],      'model\_\_dropout\_rate': [0.1, 0.2] |
| BIGRU | 21.9396 | 17.4983 | 17.5530 | 35.3950% | Lag : 24  Dropout(rate=0.1)  First BIGRU layer: 200 units  Second BIGRU layer: 100 units  Third BIGRU layer: 50 units  Fourth (last) BIGRU layer: 50 units |  |
| BIGRU-Optimized | 21.8353 | 17.6457 | 17.6187 | 34.4249% | Lag : 24,{'model\_\_dropout\_rate': 0.1, 'model\_\_bigru\_units4': 50, 'model\_\_bigru\_units3': 25, 'model\_\_bigru\_units2': 50, 'model\_\_bigru\_units1': 200} | Random Search      'model\_\_bigru\_units1': [100, 200],      'model\_\_bigru\_units2': [50, 100],      'model\_\_bigru\_units3': [25, 50, 75],      'model\_\_bigru\_units4': [25, 50],      'model\_\_dropout\_rate': [0.1, 0.2] |
| CNN | 22.5273 | 18.2384 | 17.9431 | 34.3870% | Lag : 24  Filters1=64, kernel\_size=3, activation='relu'  Dropout(0.1)  Filters2=128, kernel\_size=3, activation='relu'  Dropout(0.1)  Filters3=128, kernel\_size=3, activation='relu'  MaxPooling1D(pool\_size=2) |  |
| CNN-Optimized | 21.9288 | 17.8051 | 17.9861 | 33.7540% | Lag : 24,{'model\_\_kernel\_size2': 3, 'model\_\_kernel\_size1': 5, 'model\_\_filters2': 128, 'model\_\_filters1': 128, 'model\_\_dropout\_rate': 0.1} | Random Search      'model\_\_filters1': [32, 64, 128],      'model\_\_filters2': [64, 128, 256],      'model\_\_kernel\_size1': [3, 5],      'model\_\_kernel\_size2': [3, 5],      'model\_\_dropout\_rate': [0.1, 0.2, 0.3] |
| AutoEncoder with Best deep learning Model: LSTM | RMSE: 22.3913 | MAE: 18.2574 | MAD: 18.2574 | MAPE: 32.5959% | Lag : 24 |  |
| LSTM and optimization with Bayesian. | RMSE: 22.33 | MAE: 17.50 | MAD: 17.76 | MAPE: 37.08 | Best LSTM Configuration: {'units1': 200, 'dropout1': 0.25, 'units2': 150, 'dropout2': 0.1, 'units3': 50, 'dropout3': 0.1, 'learning\_rate': 0.000907315567349758} |  |

**Question 1:** I ran linear regression, Polynomial Regression with PCA (I used to reduce dimensions and improve performance), Moving average for Y and Moving Average with all variables, Arima for and Arima with all variables, and Gaussian process with PCA. In addition, in the plot of ACF and PACF, there is no significant seasonality.

**Question 2:** I run KNN(n\_neighbors = [3, 5, 7, 9, 11, 13, 15, 17]), Decision tree (max\_depths = [3, 5, 7, 9, 11, 13, 15, 17]) and SVR .

**Question 3:** I run GRU, LSTM, BILSTM, BIGRU, and CNN with lag 24.

**Question 4:** To tune models, I used a random search optimization. Random Search is used to efficiently explore a broad hyperparameter space for various deep learning models like GRU, LSTM, BiLSTM, BiGRU, and CNN and other machine learning models, offering a good balance between computational cost and parameter optimization.

In addition, to improve the best model of deep learning I used Bayesian Optimization.Bayesian Optimization then refines these models by targeting promising hyperparameter regions, enhancing model performance with fewer evaluations and higher precision. (All parameters are mentioned in the table)

**Question 5:**

* Statistical model: I run Stacking- With best ST models, Stacking With best ST models-PCA, Stacking with random forest, Stacking with AdaBoost. But in the end, the Best model is Arima Multivariate ARIMAX(5, 1, 5).
* Machine learning: Stacking-Best ML models (KNN, Decision tree, and SVR which are optimized with random search)

, Ensemble with Random Forrest, Stacking-Best ML models-PCA, Enhanced Stacking Model. The enhanced Stacking Model is the best model.

* Enhanced Stacking Model for machine learning algorithms: I added both a Random Forest Regressor and a Gradient Boosting Regressor as base models in the stacking ensemble. This inclusion was intended to improve the diversity and predictive power of the ensemble by incorporating models that can capture different types of patterns and dependencies in the data:

Random Forest Regressor is an ensemble learning method itself, based on bagging numerous decision trees and averaging their predictions. It is highly effective for regression and classification tasks because it reduces variance and prevents overfitting, which can be especially beneficial if some of the other models in the ensemble, like decision trees or KNN, are prone to overfitting.

Gradient Boosting Regressor builds an ensemble in a stage-wise fashion like AdaBoost but focuses more on minimizing the loss function, which is customizable and typically a gradient descent parameterization. Gradient boosting introduces new models to correct the errors made by existing ensemble members and can optimize on various loss functions, making it versatile and powerful for reducing both bias and variance in predictions.

Why Use These Models in Stacking?

Random Forest:

Strengths: Handles non-linear data well, is robust to outliers, and requires little preprocessing of data. It generally performs well across a wide range of data without the need for extensive parameter tuning.

Role in Ensemble: Provides robust and stable base-level predictions that are less likely to be influenced by noise, helping to reduce variance in the ensemble’s predictions.

Gradient Boosting:

Strengths: Highly effective at reducing bias and variance, making it one of the most powerful techniques available for improving prediction accuracy.

Role in Ensemble: Boosts the ensemble's performance by focusing on correcting the prediction errors of previous models, thus potentially improving areas where other models like SVR or decision trees might falter.

1. PCA (Principal Component Analysis): Used to reduce the dimensionality of the dataset, improving model training times and helping to prevent overfitting by removing less informative variables. PCA simplifies the complexity in high-dimensional data while retaining trends and patterns.
2. Stacking: This ensemble method combines multiple classification or regression models via a meta-classifier or meta-regressor. Stacking is used to increase the prediction accuracy by taking into account diverse perspectives from various models, leading to better generalization on unseen data.
3. Stacking with PCA: Integrating PCA into stacking reduces the input feature space, which can lead to more efficient learning and less overfitting in the stacked models. It makes the stacking ensemble not only faster but potentially more accurate by focusing on the most significant features.
4. AutoEncoder: Primarily used for unsupervised learning of efficient codings, AutoEncoders are helpful in dimensionality reduction and feature learning, especially useful in preparing data for complex predictive models by highlighting intrinsic structures or patterns in the data.
5. Bayesian Optimization: Employed for optimizing hyperparameters, Bayesian optimization uses a probabilistic model to guide the search for the best hyperparameters. This technique is particularly effective because it balances the exploration of the parameter space with the exploitation of known good regions, making it superior for finding optimal solutions efficiently.

* Deep learning: I run AutoEncoder with the Best deep learning Model: LSTM and optimization with Bayesian. But, in the end, LSTM-Optimized with random search is the best.

**Question 6:**

* Statistical model: I run linear regression, Polynomial Regression with PCA (I used to reduce dimensions and improve performance), Moving average for Y and Moving Average with all variables, Arima for and Arima with all variables, and Gaussian process with PCA. After random search optimization, the best Model is Arima Multivariate (5, 1, 5).

Because, Metrices including RMSE, MAE, MAD, and MAPE are 2.122,1.493,1.493, 2.328%respectively, and the lowest. To improve statistical models, I run Stacking- With best ST models including Linear regression, Polynomial Regression, Gaussian process with PCA, and Arima with best parameters, and Stacking- With best ST models - PCA and another with random forest. But, in the end, Arima Multivariate (5, 1, 5) is the best. The best model has metrics:

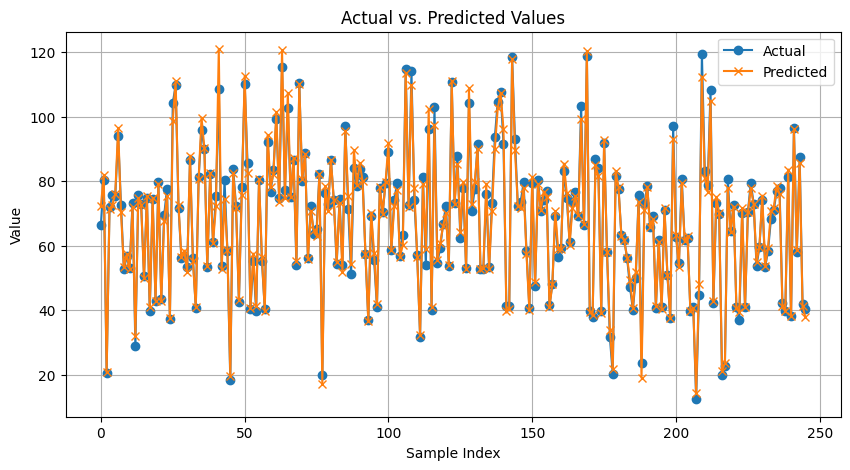
RMSE: 2.122

MAE: 1.493

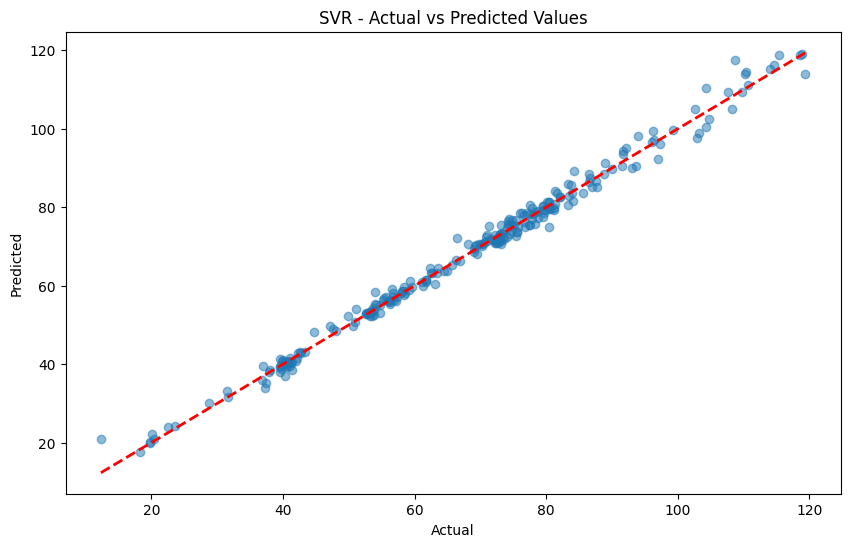
MAD: 1.493

MAPE: 2.328

And the lowest.



* Machine learning: I run KNN(n\_neighbors = [3, 5, 7, 9, 11, 13, 15, 17]), Decision tree (max\_depths = [3, 5, 7, 9, 11, 13, 15, 17]) and SVR with PCA. After random search optimization, the best model is SVR -Optimized with {'C': 3.4167643418329705, 'epsilon': 0.04115113049561093, 'gamma': 0.025764174425233172} with metrics RMSE, MAE, MAD and MAPE : 1.97,1.40, 0.98, 2.35% respectively, which is the lowest.



Then, to improve it, Stacking-Best ML models (KNN, Decision tree, and SVR which are optimized with random search)

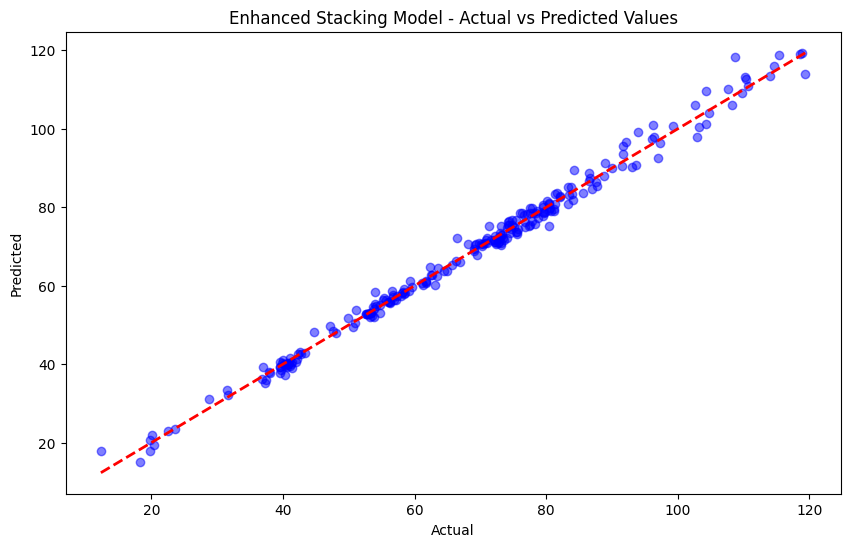
, Ensemble with Random Forrest, Stacking-Best ML models-PCA, Enhanced Stacking Model. The enhanced Stacking Model is the best model. The metrics of the best model:

RMSE: 1.93

MAE: 1.40

MAD: 1.06

Which is the lowest.



* Deep learning: I run GRU, LSTM, BILSTM, BIGRU, and CNN with lag 24. After random search optimization, the best model LSTM-Optimized has RMSE, MAE, MAD, MAPE : 21.8088,17.6202, 17.6201, 34.2585% respectively which is the lowest. Then to improve the best model, I tried AutoEncoder with the Best deep learning Model- LSTM and LSTM with parameters of optimization Bayesian. But, the best model is LSTM -Optimized with random search.

RMSE: 21.8088

MAE: 17.6202

MAD: 6201

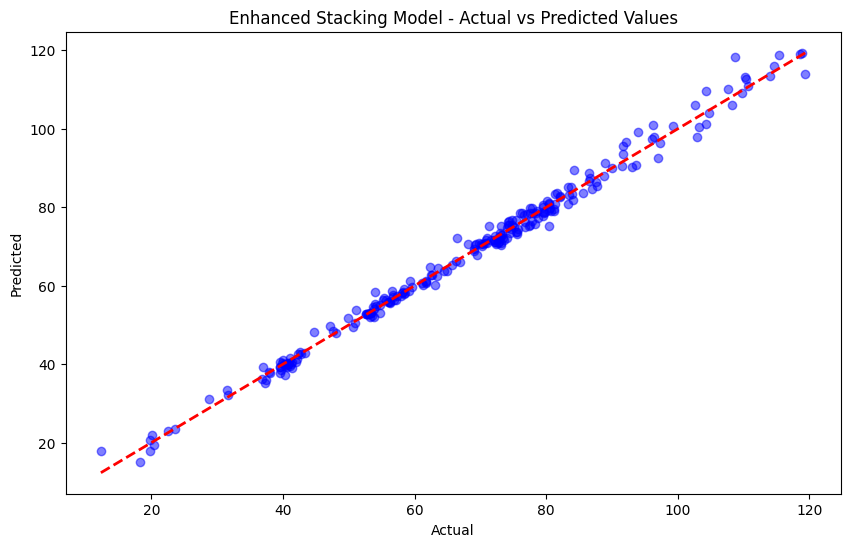
MAPE: 34.2585%

All Models: the best model is The enhanced Stacking Model is the best model (KNN, Decision tree and SVR which are optimized with random search) with performance:

RMSE: 1.93

MAE: 1.40

MAD: 1.06



**Question 7:**

* Statistical model: After random search, the worst models including Arima and Moving Average (except Arima-Multi) are not good because they have the highest metrics.
* Machine learning: After random search optimization, KNN {{'n\_neighbors': 10, 'p': 2, 'weights': 'distance'} has the highest metrics including RMSE, MAE, MAD: 2.72, 1.88, 1.22 respectively.
* Deep learning: After random search process, the worst model is CNN {'model\_\_kernel\_size2': 3, 'model\_\_kernel\_size1': 5, 'model\_\_filters2': 128, 'model\_\_filters1': 128, 'model\_\_dropout\_rate': 0.1}with the highest metrics including RMSE, MAE, MAD, and MAPE: 21.9288,17.8051,17.9861, 33.7540% respectively.

All Models: the worst model is CNN with the highest metrics including RMSE, MAE, MAD, and MAPE: 21.9288,17.8051,17.9861, 33.7540% respectively.