REPORT

Kaggle Challenge : Predicting Cyclist Traffic in Paris

Master Data Science for Business X-HEC, 2023
Schroeder Nicolas, Benslimane Mohamed (https://github.com/schronic/predicting_bike_traffic)

SUMMARY

BRIEF CASE PRESENTATION

The City of Paris's open data initiative provides a dataset related to bike usage: it contains data from bike counters across the city. The task is to use this data, along with any external datasets and to forecast the log-transformed bike count log(bike_count). The problem statement calls for a supervised regression machine learning approach, more specifically time series forecasting. Given this fact, in our analysis, as well as our feature engineering, we focused specifically on variable features, so features that have the potential to change over time (such as weather or date related data).

EXPLORATORY DATA ANALYSIS

Discovery and visualization our first dataset : train.parquet

When plotting the counters site, we see that we have 30 sites where sometimes multiple counters are installed per site. Our data is taken from 2020-09-01 01:00:00 to 2020-09-09 23:00:00. In our code, mapping counter locations in Paris and plotting site visit frequencies offers valuable spatial and frequency insights. As we will see in Fig 3, this visualization helps in understanding which sites are more popular.

	counter_id	counter_name	site_id	site_name	bike_count	date	counter_installation_date	coordinates	counter_technical_id	latitude	longitude	log_bike_count
48321	100007049-102007049	28 boulevard Diderot E-O	100007049	28 boulevard Diderot	0.0	2020-09-01 02:00:00	2013-01-18	48.846028,2.375429	Y2H15027244	48.846028	2.375429	0.000000

Fig 1: Discovering our features through the head of the given dataset

In the next phase of our analysis, we shifted focus to identifying the most frequented sites, as this was key to understanding the factors influencing the total 'bike_count'. An exemplifying case is the location "Totem 73 boulevard de Sébastopol,"the location with highest bike traffic, shown in Fig 3. As expected, on a macro level, there are significant peaks in ridership during the summer as we see on Fig 2. On a daily basis we see strong cyclic patterns that seem to repeat themselves throughout the year. We see few rides during the night, as well as a stark increase during rush hour times. (Fig 4)

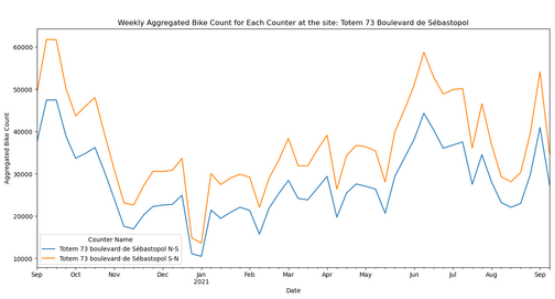


Fig 2

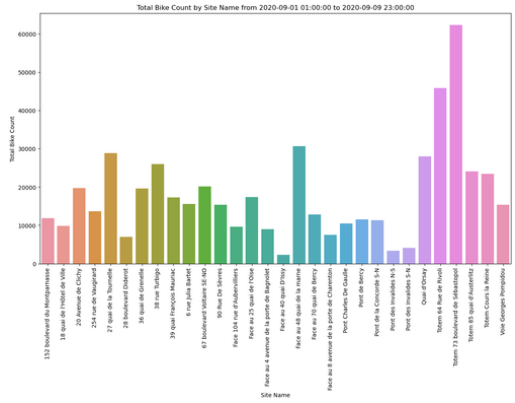


Fig 3

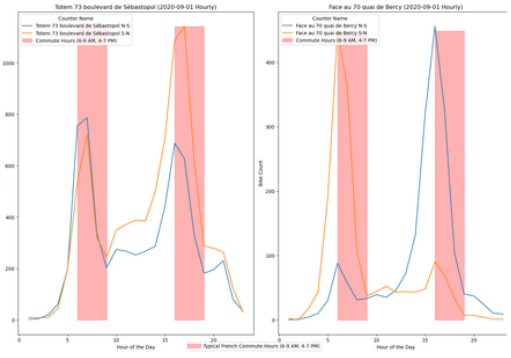


Fig 4

Skewed distribution of the target variable

Upon plotting the 'bike_count' variable on a daily basis, following the hints on the given notebook and Fig 4, we realized that the variable exhibited a skewed behavior. This skewness posed a challenge because it indicated a non-normal distribution of the data. In statistical modeling and analysis, normal distribution of variables is often a key assumption, especially in linear models. A skewed distribution, with its asymmetry and potential for extreme values, can lead to biases in model predictions and affect the accuracy of the analysis.

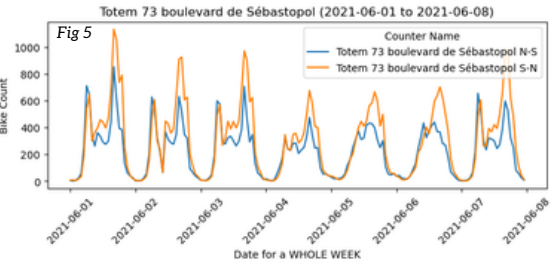


Fig 5

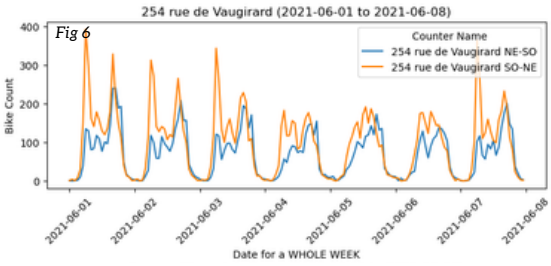


Fig 6

These two graphs indicate that the bike_count data is highly skewed to the right, with a large number of counts close to zero and few counts reaching higher values. Fig 4 also confirms this skewness by showing the peak near the lower end of the bike_count range and a long tail stretching towards the higher counts.

Recognizing this issue is crucial as it guides the subsequent steps in data processing and model selection, ensuring that the methodologies we employ are appropriate for the nature of the data. In theory, the choice of ML models are significantly impacted by the skewed distribution of the initial target variable, for example, if the original data is skewed, some models may not perform well in theory, and more robust or non-parametric models might be needed. So to avoid potential errors or misunderstanding, in the following, we use `log_bike_count` as the target variable as otherwise we will be restricted with `bike_count` that would be sometimes inappropriate to model due to it's distribution for the above given reasons.

The stationarity of the new target

As in a lot of time series forecasting models stationarity of the target variable is assumed. Hence we turned to the Dickey Fuller test, which confirmed the stationary nature of our variable. The results of the Dickey-Fuller test were quite telling, with a p-value significantly lower than our significance level of 5%. This allowed us to confidently reject the null hypothesis, that the 'log_bike_count' time series is non-stationary and contains a unit root.

Fig 7: Dickey Fuller Test

```
Results of Dickey-Fuller Test:
Test Statistic                -1.183813e+01
p-value                        7.703468e-22
#Lags Used                     2.000000e+00
Number of Observations Used    4.970000e+02
Critical Value (1%)           -3.443576e+00
Critical Value (5%)           -2.867373e+00
Critical Value (10%)          -2.569877e+00
dtype: float64
```

Why is this finding important and beneficial? **(1) Consistency in Mean and Variance:** The stationarity of a time series implies consistent mean, variance, and autocorrelation over time. This consistency is advantageous for developing models that can accurately predict future values, as these predictions are based on the premise that future statistical properties will reflect those of the current series. **(2) Long-term Forecasting:** Stationary data typically does not exhibit long-term trends, enhancing the reliability of long-term forecasting. Predictions made under stationary conditions are less likely to be influenced by evolving trends over time. In summary, it validates our approach and allows us to confidently proceed with time series analysis, modeling, and forecasting.

Handling null values

In our dataset, while there are no explicit missing values, there's a significant portion (about 1%) of values for 'log_bike_count' that are zero, throughout stretches of over a day, for certain counters,. We chose to illustrate this by examining two specific sites above: "152 Boulevard du Montparnasse" and "20 Avenue de Clichy". The results from this investigation revealed a real loss of information: we observed null values for 20 days at the Montparnasse site and for 3 months at Avenue Clichy.

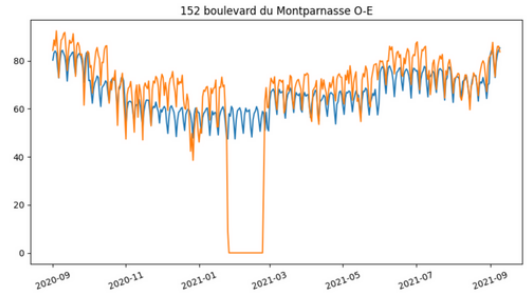
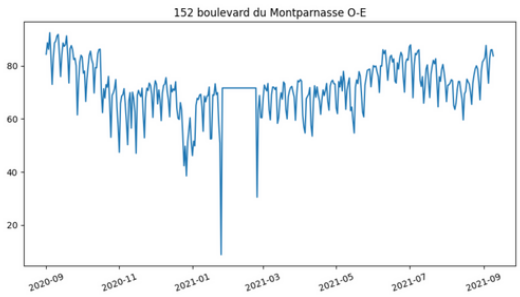


Fig 8: Possible option to predict the null values

Fig 9: Possible option to replace the null values by the mean of the target for the whole counter



While the reasons for this can be manifold, such as malfunction or the simple choice to turn them off, these issues pose the risk of negatively influencing our models predictive capabilities. To address these null values, we considered four different approaches. (1) Replace them by the mean for each counter. (2) Predict them using a model to estimate the missing values. (3) Not doing anything - an option that entails leaving the dataset as is, without any intervention for the null values, or lastly (4) deleting them. Given that we are not talking about a missing covariate, but missing target, **it seemed most appropriate to delete these values**. Nonetheless we will see in the modelization part that we decided to run the models with both datasets to compare results.

FEATURE ENGINEERING AND DATA PREPROCESSING

Feature Selection and Data Cleaning

1) Feature Selection and Data Cleaning on the given train.parquet: Initially, we scrutinized the correlation matrix, and even considered an embryonic stage of Principal Component Analysis (PCA), though our mastery of PCA isn't yet perfect, we intended to include it in our code to eventually reduce the dimensionality and prevent overfitting. The heatmap revealed no particularly strong correlations among the different variables. The absence of strong correlations among the other variables indicated that multicollinearity, which can adversely affect certain models, was not a significant issue for this dataset, at least in the numeric features. For “site_id” and “counter_id” variables we expect a strong correlation, as 56 counters were placed at 30 sites. Consequently, there was no immediate need to remove any explanatory variables based on correlation analysis alone for the train.parquet. When it comes to cleaning the outliers we decided to keep these outliers after the Interquartile Range (IQR) method.

(2) Feature Selection and Data Cleaning on the New Paris weather dataset: Since the external_data.csv file contained weather data only at three hour granularity, we decided to find our own dataset with **more granular data**, as the disparity in data resolution could potentially result in misaligned insights and, as a consequence, less accurate predictions. This new file is named "Paris 2020-09-01 to 2021-10-31.csv" in our data folder.

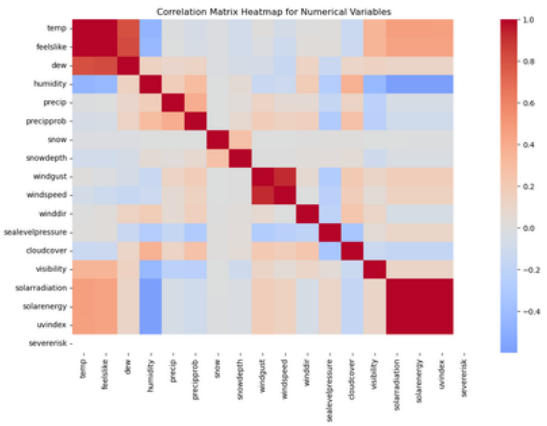


Fig 11: Heatmap for the new weather dataset

Fig 10: Finding outliers in the new dataset with the Interquartile Range (IQR) method

```
temp - Outliers: 18 (0.18%)
feelslike - Outliers: 32 (0.31%)
dew - Outliers: 57 (0.56%)
humidity - Outliers: 15 (0.15%)
precip - Outliers: 1432 (14.00%)
precipprob - Outliers: 1432 (14.00%)
snow - Outliers: 1 (0.01%)
snowdepth - Outliers: 24 (0.23%)
windgust - Outliers: 158 (1.55%)
windspeed - Outliers: 123 (1.20%)
winddir - Outliers: 0 (0.00%)
sealevelpressure - Outliers: 275 (2.69%)
cloudcover - Outliers: 0 (0.00%)
visibility - Outliers: 587 (5.74%)
solarradiation - Outliers: 1055 (10.32%)
solarenergy - Outliers: 1001 (9.79%)
uvindex - Outliers: 591 (5.78%)
severerisk - Outliers: 0 (0.00%)
```

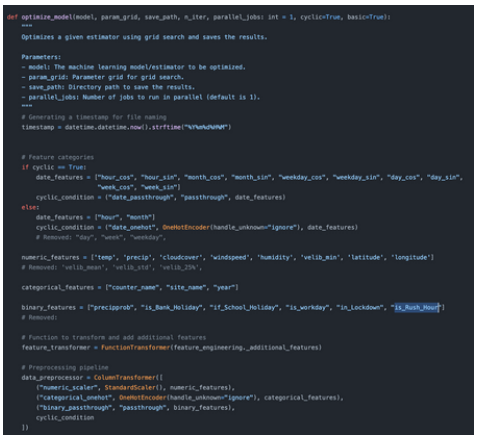
In the new weather dataset, we encountered variables with excessive missing values, which we opted to remove outright, since we also assumed these features to have little impact on bike ridership. Secondly, we decided to retain variables with outliers, even those with as high as 14% outliers, like the 'precipprob' variable as we see in Fig 10, because we found that these were not measurement errors, and hence might carry value for our predictions. The correlation matrix heatmap (Fig 11) provided a clear visual representation of the relationships between various meteorological variables. We observed that certain variables, such as 'solarradiation' and 'uvindex', exhibited a high degree of correlation. After removing natural choices of features, the issue of collinearity resolved itself. As a last step we scaled weather data using aMinMaxScaler. All theses steps gave us `scaled_weather_data.csv` (Fig 12).

	datetime	temp	humidity	precip	precipprob	windspeed	cloudcover
0	2020-09-01 00:00:00	0.512563	0.631085	0.0	0.0	0.150000	0.333
1	2020-09-01 01:00:00	0.497487	0.677387	0.0	0.0	0.152174	0.100

Fig 12: The head of the new weather dataset after feature selection

Feature Transformation, Construction, Encoding and Merging our final dataset

Let's focus now on giving examples of how we created new variables and encoding variables, as the two below examples are not the only encoding we did, we invite you to see our repository to see all the new variables we created.

Examples of encoding	*Methodology*	*How we code it*
Example 1: Encoding with OneHotEncoder Objective: The primary objective is to appropriately transform categorical features into a machine-learning-friendly format using one-hot encoding, ensuring these features can be effectively utilized in model training and prediction.	(1) We identify all categorical features in the given and new datasets that require encoding. (Example: counter_name). (2)We employ the OneHotEncoder to convert each unique value of a categorical feature into a new binary (0 or 1) feature. (3)One-hot encoding is integrated into a broader preprocessing pipeline, which may also include scaling for numerical features and other transformations. The ColumnTransformer from Scikit-learn is used to apply different transformations to different columns: one-hot encoding for categorical features and scaling for numerical features.	
Example 2: Calculating and Encoding Distances in Velib Data Dataset: Velib station data alongside another dataset with location coordinates. Objective: Compute and encode the distances between Velib stations and various points of interest represented in the secondary dataset. This encoding is vital for understanding the spatial distribution and accessibility of Velib stations.	We employed geospatial distance calculations using geopy.distance. This method allowed us to precisely measure the distance between each Velib station and the coordinates provided in the secondary dataset. A distance matrix was created where each row represented a Velib station, and each column contained the distance to a different location.	

Merging: The final dataset is obtained by merging these transformed and processed datasets. The merge process involves aligning the datasets on common keys or indices (likely timestamps and locations) and consolidating the features from each dataset into a unified structure.

MODEL PIPELINE

Model Selection Rationale

In this project, we evaluated a range of models, including LightGBM, XGBoost, MLP Network, ElasticNet, and Ridge Regression, due to their proven applicability in time series forecasting. We decided against auto regressive models, a popular choice for timeseries prediction, as the data exhibited significant non-linear patterns and complex interactions, which these models are typically not well-suited to capture. The use of gradient boosting models like LightGBM and XGBoost was influenced by their efficiency in handling large datasets and their capacity for feature importance evaluation, which is critical in understanding the key drivers of bike ridership trends. The inclusion of the MLP Network, a form of deep learning, was motivated by its ability to capture complex, nonlinear relationships in data, which can be significant in urban mobility contexts. ElasticNet and Ridge regression were chosen for their robustness in handling multicollinearity and for providing a balance between feature selection and regularization, which is important in a dataset with many potential predictor variables.

Hyperparameters tuning and Cross-Validation Strategy

For hyperparameter tuning and validation, we employed a combination of feature transformation and cross-validation techniques tailored for time series data. The data preprocessing steps included scaling of numeric features, one-hot encoding of categorical variables, and passthrough for binary features. Cyclic date features were treated both via one-hot encoding and cyclic encoding. For each model results were compared. While one-hot encoding is easily interpretable by most models, including tree-based algorithms, it can lead to dimensionality issues. Cyclic encoding on the otherhand (using sine and cosine) captures the continuous and cyclical nature of time, thereby preserving information about the proximity of different time periods. Even though in theory the latter can be problematic for tree-based models, as these models may struggle to interpret the continuous, circular relationships represented in this form, especially given that information is split into two columns, in our case cyclic encoding for the XGBoost Regressor outperformed one-hot encoding.

We utilized a TimeSeriesSplit with eight splits for cross-validation to ensure that the temporal sequence of the data is maintained, which is critical for time series forecasting. The training and testing sets were selected through a randomized sampling approach, aiming to capture a representative yet diverse range of data patterns. The model pipeline integrated the feature transformer and data preprocessor with the respective model.

RandomizedSearchCV was used for hyperparameter optimization, choosing parameters randomly from a specified grid, which is more efficient compared to exhaustive grid search, especially given the computational complexity. A large part of our iterative working process was the consistent monitoring and adjustment of the parameter search grid. The search was conducted with the negative root mean squared error (negative RMSE) as the scoring metric, ensuring a focus on minimizing prediction errors.

colsample_bylevel	0.504144
colsample_bynode	0.537176
colsample_bytree	0.946781
gamma	4.595150
learning_rate	0.101262
max_depth	32.000000
n_estimators	238.000000
reg_alpha	0.771622
reg_lambda	0.386605
subsample	0.777861

Fig 13 : Output of crossvalidation

Computational Considerations

Our computational setup was based on an macOS Monterey 12.1 with Apple M1 Chip 8- 809 Core CPU. Given the limitation in computational resources, we opted for RandomizedSearchCV over GridSearchCV due to its reduced computational complexity. To further optimize computational efficiency, we tried to minimize redundancies in data processing and heavily relied on existing, optimized libraries. The use of n_jobs=-1 allowed for parallel processing, significantly reducing the computation time in model training.

MODEL EVALUATION

Our analysis revealed that the dataset with removed NaN values in the target variable outperformed the complete set. The XGBoost model emerged as the best performer (RMSE 0.68), closely followed by the LightGBM (RMSE 0.70) and MLP Network (RMSE 0.69) model.

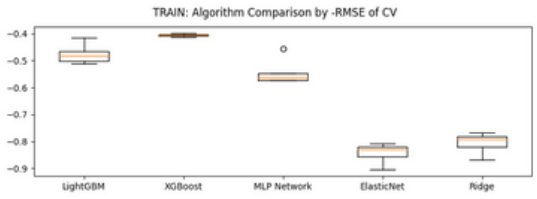


Fig 14

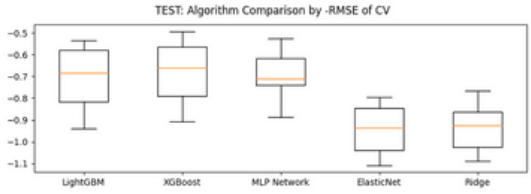
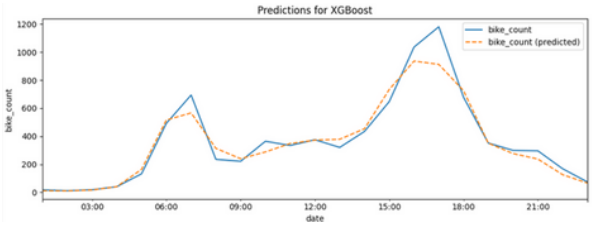


Fig 15

All three models were prone to overfitting, which we addressed through careful hyperparameter tuning. This tuning included adjusting tree depth, number of estimators or layers, and introducing alpha and lambda regularization. Feature reduction was also a key part of our strategy to combat overfitting. Specifically, we decided to drop all "velib" related data, as these did not seem to have great importance in our model accuracy, and by construction were not able to explain any of the variable fluctuation in bike ridership over time.

Fig 15



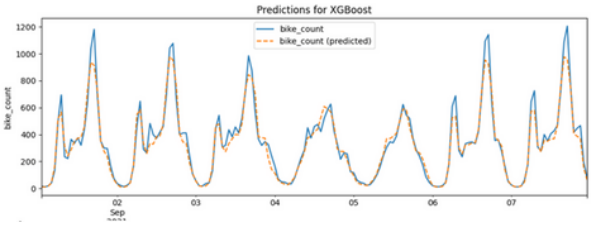


Fig 16

Comparison to Baselines and Previous Models

Compared to a simple mean baseline, our models showed significant improvements. However, when compared to the Ridge regression baseline, the extent of outperformance was modest. This suggests that while our models were effective in capturing complex patterns in the data, there is still room for improvement, especially in terms of capturing subtle nuances that simpler models like Ridge regression can miss.

	model	mean_train_score	std_train_score	mean_test_score	std_test_score	mean_fit_time	std_fit_time	mean_score_time	std_score_time
0	LightGBM	-0.476283	0.029887	-0.702731	0.135517	10.302388	4.830360	0.815385	0.093611
1	XGBoost	-0.405279	0.004314	-0.675963	0.135330	6.940925	3.793191	0.746635	0.174024
2	MLP Network	-0.549621	0.037298	-0.694278	0.113551	38.770833	17.840814	0.232660	0.035455
3	ElasticNet	-0.842425	0.031390	-0.942509	0.112491	2.968295	1.650053	0.106156	0.015700
4	Ridge	-0.804171	0.033178	-0.932879	0.102874	1.146791	0.555068	0.097898	0.013810

Fig 18

Our final training data, after encoding, had 222 features. The top 10 features, ranked by "gain", a metric that quantifies the importance of a feature via the frequency with which it appears and contribution it has on the overall prediction error can be found in Figure

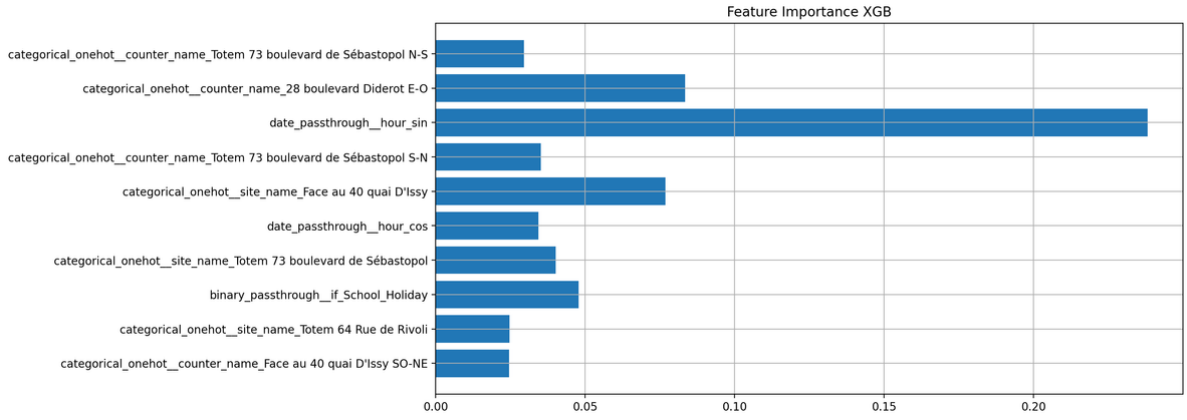


Fig 19

CONCLUSION

Our analytical efforts in this project have indicated that models adept at handling complex, non-linear relationships—specifically XGBoost, LightGBM, and MLP Network—outperform those based on more traditional statistical approaches in predicting bike ridership. By far the most important feature in our case is "hour" data, followed by location specific data. While we have worked extensively on the issue of overfitting data, our efforts have only gotten us so far as the issue continues to persist. The project underscored the importance of meticulous feature selection. The decision to remove NaN values and unrelated data, without predictive significance, proved as crucial decisions. This work has highlighted the delicate balance between model complexity and interpretability, especially when juxtaposed against simpler models like Ridge regression, which, despite their modesty, maintain relevance in understanding the data's underlying structure.