Celik Ennis
Departement Electronics ICT - Applied Artificial Intelligence
Thomas More Mechelen-Antwerp - Campus De Nayer

May 31, 2025

REPORT — Celik E.

Contents

1 Introduction

1.1 Course Overview

The Python for AI course provided a comprehensive introduction to the fundamentals of Python programming, focusing on its applications in artificial intelligence and machine learning. The course covered essential topics such as data manipulation (Pandas), statistical analysis (Matplotlib , Pandas), data-preprocessing (Pandas , Numpy , Scikit-learn) and model training (Tensorflow , Scikit-learn).

1.2 Final Project Overview

The final project aimed to apply the knowledge and skills acquired throughout the course to a dataset of choice. The project involved selecting a dataset, performing data preprocessing, feature engineering, model training, evaluation and reporting back our results (hence this report). The goal was to build a machine learning model that could effectively analyze the dataset and provide meaningful insights.

1.3 Dataset Selection

The dataset chosen for the project was the "StudentPerformanceFactors" dataset from Kaggle. This dataset contains various factors that may influence student performance, including demographic information, study habits, and academic records.

```
see => https://www.kaggle.com/datasets/larsen0966/student-performance-data-set

Attribute                       Description
Hours_Studied                   Number of hours spent studying per week.
Attendance                      Percentage of classes attended.
Parental_Involvement            Level of parental involvement in the student's education (Low, Medium, High).
Access_to_Resources             Availability of educational resources (Low, Medium, High).
Extracurricular_Activities      Participation in extracurricular activities (Yes, No).
Sleep_Hours                     Average number of hours of sleep per night.
Previous_Scores                 Scores from previous exams.
Motivation_Level                Student's level of motivation (Low, Medium, High).
Internet_Access                 Availability of internet access (Yes, No).
Tutoring_Sessions               Number of tutoring sessions attended per month.
Family_Income                   Family income level (Low, Medium, High).
Teacher_Quality                 Quality of the teachers (Low, Medium, High).
School_Type                     Type of school attended (Public, Private).
Peer_Influence                  Influence of peers on academic performance (Positive, Neutral, Negative).
Physical_Activity               Average number of hours of physical activity per week.
Learning_Disabilities           Presence of learning disabilities (Yes, No).
Parental_Education_Level        Highest education level of parents (High School, College, Postgraduate).
Distance_from_Home              Distance from home to school (Near, Moderate, Far).
Gender                          Gender of the student (Male, Female).

Exam_Score                      Final exam score.
```

My objective is to see/analyze which features/factors influence the students exam score the most.

1.4 Goal

My goal for this project is to analyze the dataset and build a machine learning model that can predict the students exam score based on the various factors provided in the dataset. And to at least achieve a MSE close to 0 with a margin of 0.2 .

2 Project

2.1 Data Exploration and Preprocessing

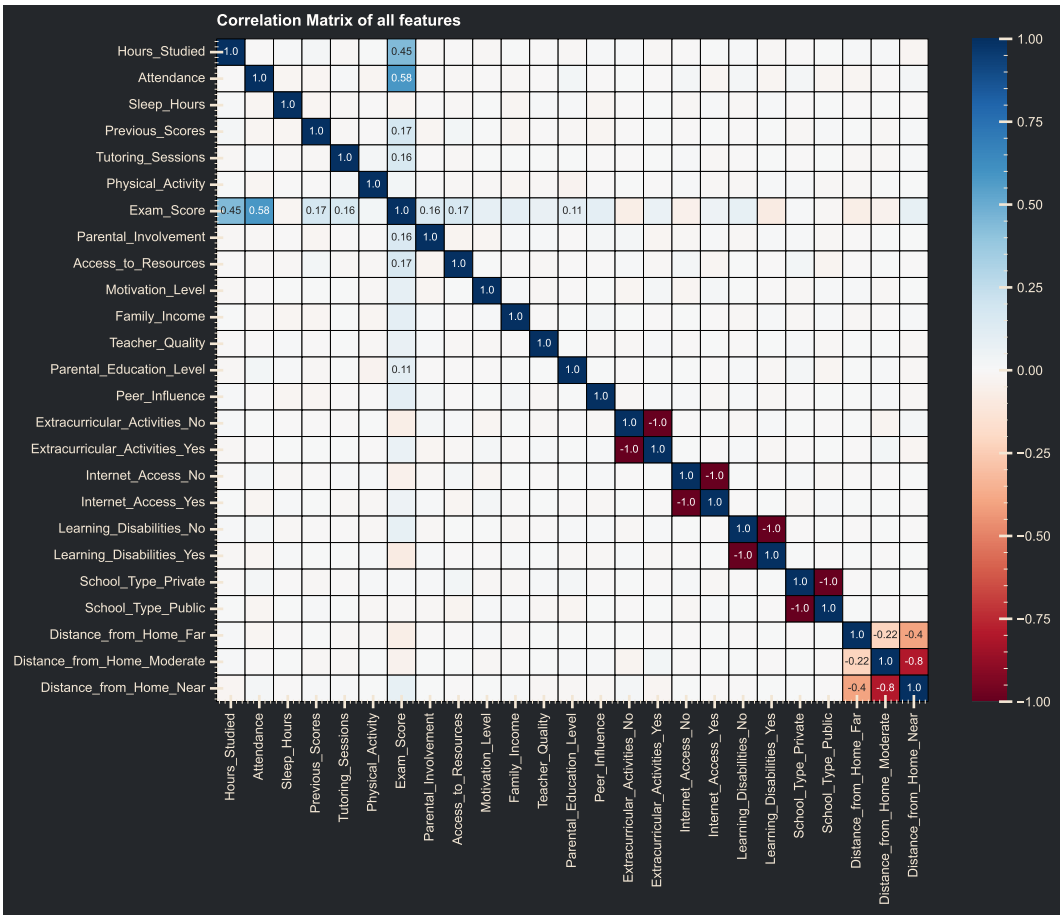
2.1.1 Exploration

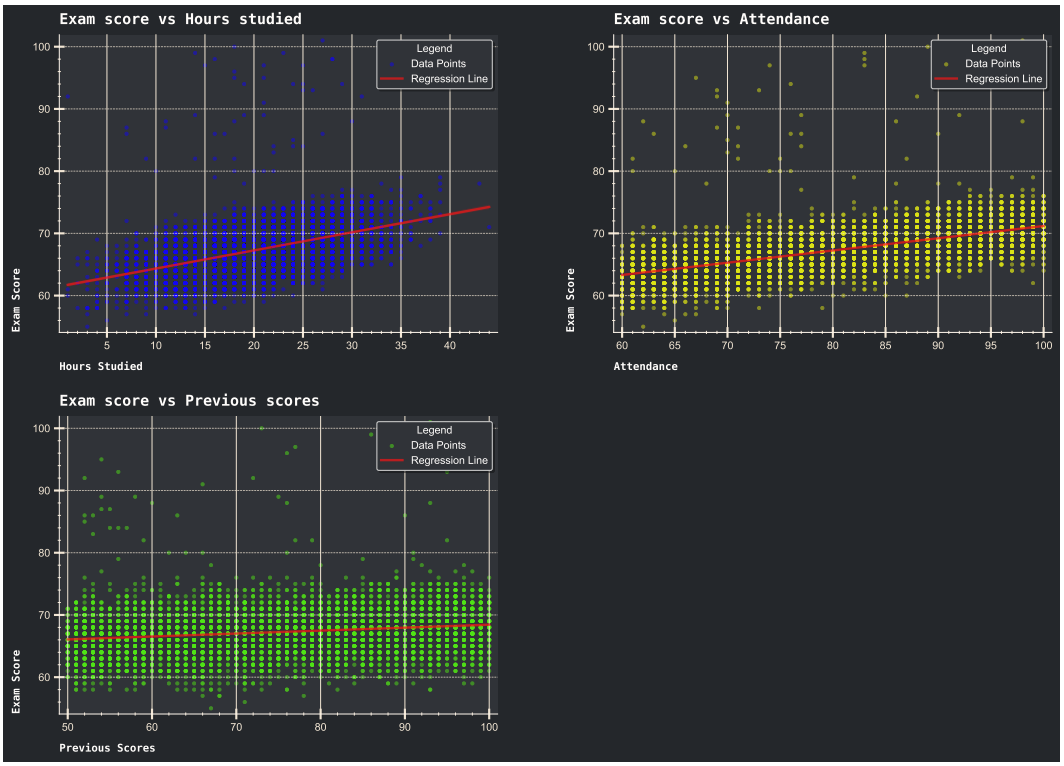
The first step in the project was to explore and preprocess the dataset. We explore these 2 steps in chapter 1 and 2 of the code. This involved loading the dataset, examining its structure, and identifying any missing or inconsistent data. After that, we performed data preprocessing to clean the dataset and prepare it for analysis and pass it on to a ML model. In this case, the dataset was relatively clean, with a few missing values here and there and some exam scores outside of the normal $[0; 100]$ range.

```
#  #   Column                      Non-Null Count  Dtype
# ---  ------                      --------------  -----
#  0   Hours_Studied               6607 non-null   int64
#  1   Attendance                  6607 non-null   int64
#  2   Parental_Involvement        6607 non-null   object
#  3   Access_to_Resources         6607 non-null   object
#  4   Extracurricular_Activities  6607 non-null   object
#  5   Sleep_Hours                 6607 non-null   int64
#  6   Previous_Scores             6607 non-null   int64
#  7   Motivation_Level            6607 non-null   object
#  8   Internet_Access             6607 non-null   object
#  9   Tutoring_Sessions           6607 non-null   int64
#  10  Family_Income               6607 non-null   object
#  11  Teacher_Quality             6529 non-null   object
#  12  School_Type                 6607 non-null   object
#  13  Peer_Influence              6607 non-null   object
#  14  Physical_Activity           6607 non-null   int64
#  15  Learning_Disabilities       6607 non-null   object
#  16  Parental_Education_Level    6517 non-null   object
#  17  Distance_from_Home          6540 non-null   object
#  18  Gender                      6607 non-null   object
#  19  Exam_Score                  6607 non-null   int64
```

```
#        Hours_Studied   Attendance  Sleep_Hours  Previous_Scores  Tutoring_Sessions   Exam_Score
# count    6607.000000  6607.000000   6607.00000      6607.000000        6607.000000  6607.000000
# mean       19.975329    79.977448      7.02906        75.070531           1.493719    67.235659
# std         5.990594    11.547475      1.46812        14.399784           1.230570     3.890456
# min         1.000000    60.000000      4.00000        50.000000           0.000000    55.000000
# 25%        16.000000    70.000000      6.00000        63.000000           1.000000    65.000000
# 50%        20.000000    80.000000      7.00000        75.000000           1.000000    67.000000
# 75%        24.000000    90.000000      8.00000        88.000000           2.000000    69.000000
# max        44.000000   100.000000     10.00000       100.000000           8.000000   101.000000
```

As we explore further into dataset by drawing up the correlation matrix, we can see that some features are highly correlated with the exam score, such as the number of hours studied or the attendance. (the picture below shows the correlation matrix of the dataset keep in mind that the correlation matrix is not a perfect representation of the data, but it gives us a good idea of which features are correlated with the exam score)





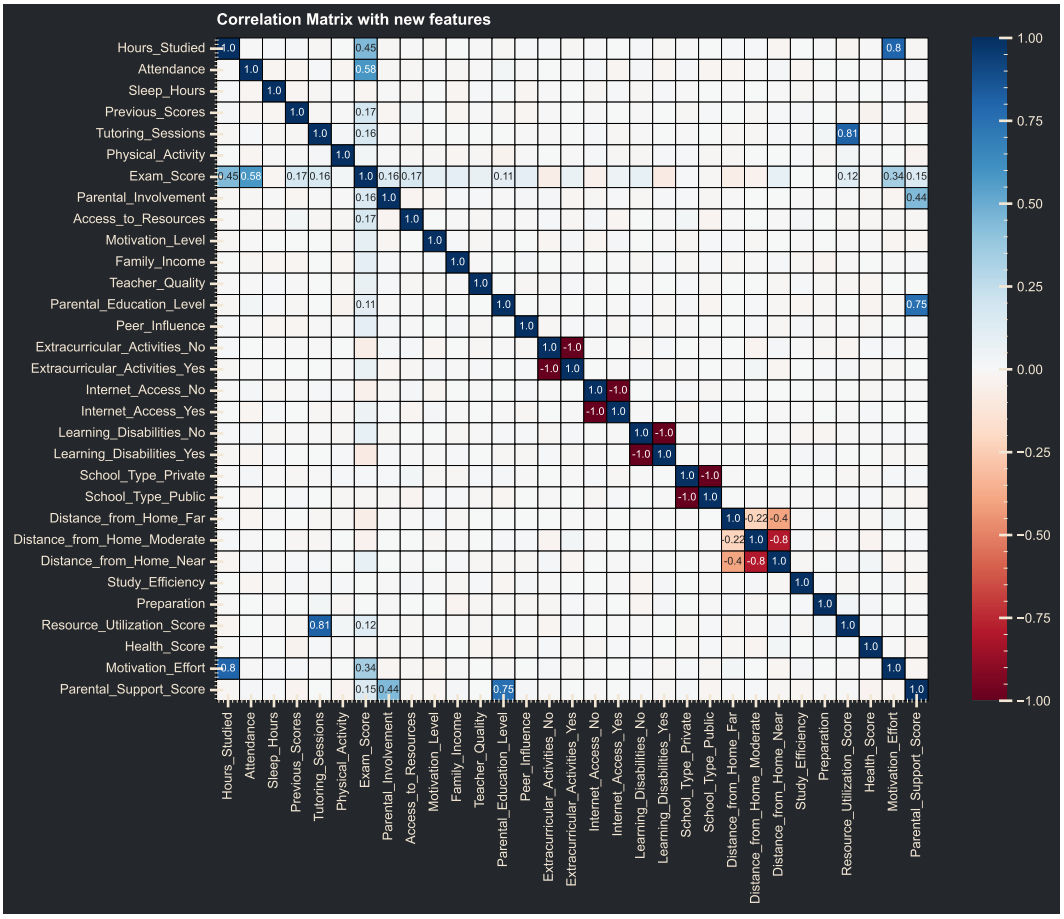
I did try to create new features:

1. Study Efficiency : a combination of the number of hours studied and the attendance rate, to see if students who study more and attend more classes perform better.
2. Preparation : a combination of tutoring sessions and previous exam scores, to see if students who have more tutoring sessions and better previous exam scores perform better.
3. Resource Utilization Score : a combination of Access to Resources, Internet Access and Tutoring sessions

to see if students which different eductional resources would perform better.

4. Health Score : a combination of sleep hours, physical activity to see if students who take care of their health perform better.
5. Motivation Effort : a combination of motivation and hours studied to see if students who are more motivated and put in more effort perform better.
6. Parental Support Score : a combination of parental involvement and parental education level to see if students who have more parental support perform better.

These new features were created to see if they would improve the performance of the model, but in the end they did not improve the performance of the model. As you will see later on. The correlation matrix however show that some of these features do have some correlation with the exam score, but not enough to make a significant difference in the performance of the model.



Now again keep in the mind the correlation matrix only shows linear correlation, so it is possible that some of these features do have a non-linear correlation with the exam score.

2.1.2 Preprocessing

The next step was to preprocess the dataset. This involved handling missing values, encoding categorical variables, and scaling numerical features. I first cleaned the dataset by removing any rows with missing values and then I encoded the categorical variables using one-hot encoding or ordinal encoding depending on the categorical column. For example , categories like "yes" and "no" were encoded as 1 and 0 respectively, while categories like "High", "Medium", "Low" were encoded as 4, 3, 2, 1 respectively.

```
# Parental_Involvement                : ['High' 'Low' 'Medium']
# Access_to_Resources                 : ['High' 'Low' 'Medium']
# Extracurricular_Activities          : ['No' 'Yes']
# Motivation_Level                    : ['High' 'Low' 'Medium']
# Internet_Access                     : ['No' 'Yes']
# Family_Income                       : ['High' 'Low' 'Medium']
# Teacher_Quality                     : ['High' 'Low' 'Medium']
# School_Type                         : ['Private' 'Public']
# Peer_Influence                      : ['Negative' 'Neutral' 'Positive']
# Learning_Disabilities               : ['No' 'Yes']
# Parental_Education_Level            : ['College' 'High School' 'Postgraduate']
# Distance_from_Home                  : ['Far' 'Moderate']
```

For numerical features , i used a StandardScaler to scale the features to have a mean of 0 and a standard deviation of 1. Just to be sure we also use a simple imputer to fill in any missing values with the mean of the column , or the most frequent value. Putting it all together, inside a Pipeline and a ColumnTransformer to apply the preprocessing steps to the numerical and categorical features.

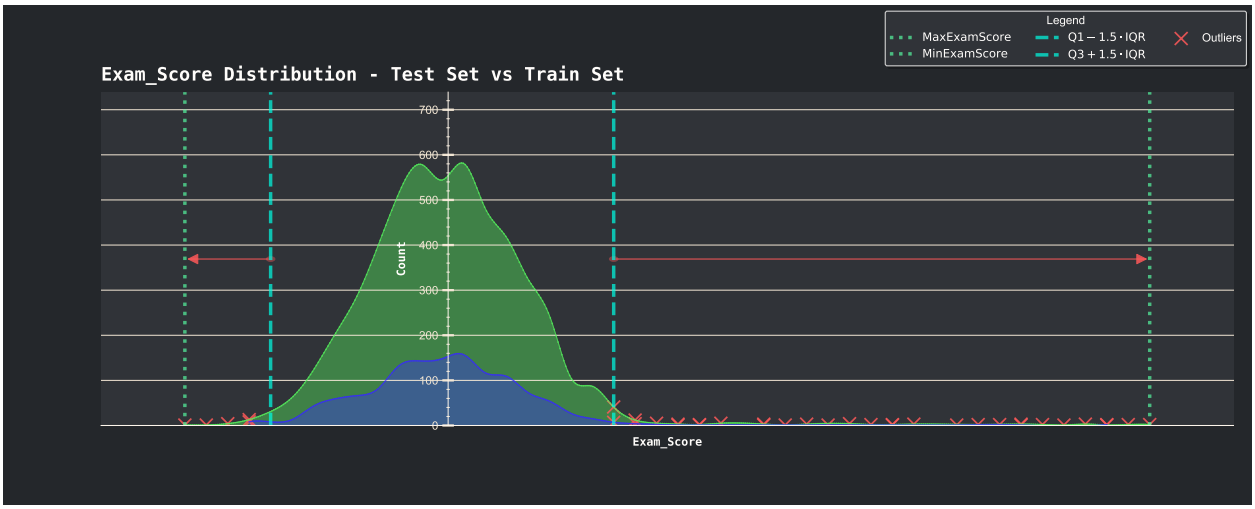
```
numerical_transformer = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="mean")),
        ("scaler", StandardScaler()),
    ]
)
categorical_transformer_ordinal = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("encoder", OrdinalEncoder(categories=ordinal_categories)),
    ]
)
categorical_transformer_nominal = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("encoder", OneHotEncoder(handle_unknown="ignore")),
    ]
)

ct = ColumnTransformer(
    transformers=[
        ("numerical_cols",numerical_transformer,numerical_cols),
        ("ordinal_categories",categorical_transformer_ordinal,ordinal_cols),
        ("nominal_categories",categorical_transformer_nominal,nominal_cols),
    ]
)
```

We then applied the preprocessing steps to the dataset that has been split it into training and testing sets. And finally we saved the preprocessed dataset to a CSV file for later use. Also keeping a copy of the original dataset for reference and saving a test and train dataset with the columns i want to keep.

```
    # Hours_Studied                 0.445104 <-
    # Attendance                    0.580259 <-
    # Previous_Scores               0.174283 <-
    # Tutoring_Sessions             0.156829 <-
    # Parental_Involvement          0.156014 <-
    # Access_to_Resources           0.167856 <-
    # Parental_Education_Level      0.105253 <-
    # Resource_Utilization_Score    0.121853 <-
    # Motivation_Effort             0.338626 <-
    # Parental_Support_Score        0.145135 <-
```

resulting in this exam score distribution plot:



However we can see that the exam score distribution is not normal and has some outliers, which can affect the performance of the model. So I decided to remove the outliers from the dataset by removing any rows with exam scores outside of the range $[Q1 - 1.5IQR, Q1 + 1.5IQR]$. resulting in this exam score distribution plot:


Legend
MaxExamScore
MinExamScore
Q1 − 1.5 · IQR
Q3 + 1.5 · IQR
Exam_Score Distribution - Test Set vs Train Set
700
600
500
400
300
200
100
0
Count
Exam_Score

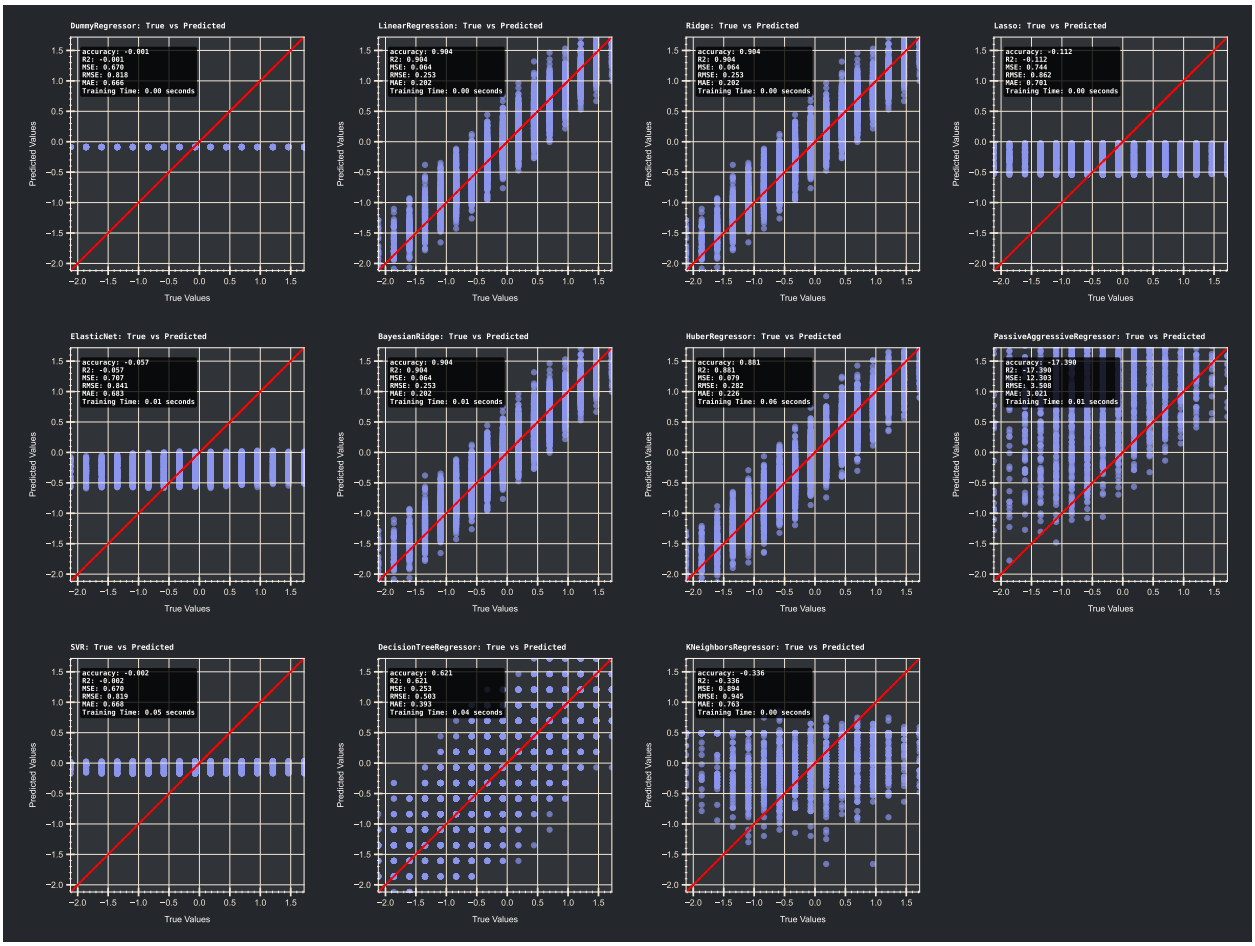
2.2 Searching for models

2.2.1 Model Selection

After preprocessing the dataset, the next step was to select a machine learning model to train on the dataset. I decided to use a regression model since the goal is to predict the exam score, which is a continuous variable. I started with a simple linear regression model, and other ML models such as KNNRegressor,SVR,etc... , which is a good baseline model for regression tasks. Alongside i used Cross-Validation to evaluate the performance of the model on the training set.

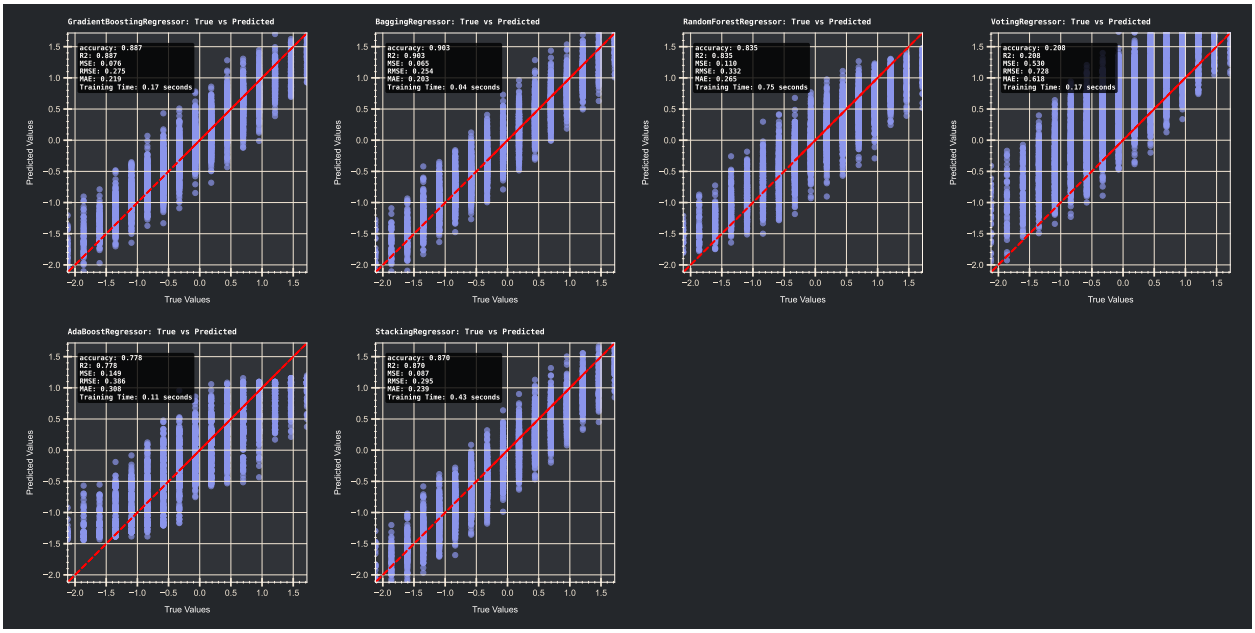
```
models = {
    "DummyRegressor" : DummyRegressor(),
    "LinearRegression" : LinearRegression(),
    "Ridge" : Ridge(random_state=RANDOM_SEED),
    "Lasso" : Lasso(random_state=RANDOM_SEED),
    "ElasticNet" : ElasticNet(random_state=RANDOM_SEED),
    "BayesianRidge" : BayesianRidge(),
    "HuberRegressor" : HuberRegressor(),
    "PassiveAggressiveRegressor" : PassiveAggressiveRegressor(random_state=RANDOM_SEED),
    "SVR" : SVR(),
    "DecisionTreeRegressor" : DecisionTreeRegressor(random_state=RANDOM_SEED),
    "KNeighborsRegressor" : KNeighborsRegressor(),
}
```

some of these models performed well on the test set, but others did not perform well at all.



	Model	Cross-Validation		Metrics					Time
index	Model	cv_mean_r2	cv_std_r2	accuracy	R2	MSE	RMSE	MAE	training_time
0	DummyRegressor	-0.008	0.005	-0.001		0.67	0.818	0.666	0.001
1	LinearRegression	0.897	0.019	0.904	0.904	0.064	0.253	0.202	0.004
2	Ridge	0.897	0.019	0.904	0.904	0.064	0.253	0.202	0.003
3	Lasso	-0.006	0.006	-0.112		0.744	0.862	0.701	0.003
4	ElasticNet	0.048	0.01	-0.057		0.707	0.841	0.683	0.005
5	BayesianRidge	0.897	0.019	0.904	0.904	0.064	0.253	0.202	0.007
6	HuberRegressor	0.88	0.02	0.881	0.881	0.079	0.282	0.226	0.057
7	PassiveAggressiveRegressor	-0.925	3.065	-17.39		12.303	3.508	3.021	0.008
8	SVR	-0.016	0.013	-0.002		0.67	0.819	0.668	0.053
9	DecisionTreeRegressor	0.622	0.077	0.621	0.621	0.253	0.503	0.393	0.043
10	KNeighborsRegressor	-0.248	0.148	-0.336		0.894	0.945	0.763	0.004

I then tried some more advanced models such as RandomForestRegressor, BaggingRegressor, StackingReggressor. Essentially pulling out all the ensemble methods I could find in the scikit-learn library.



	Model	Cross-Validation		Metrics					Time
index	Model	cv_mean_r2	cv_std_r2	accuracy	R2	MSE	RMSE	MAE	training_time
0	GradientBoostingRegressor	0.875	0.018	0.887	0.887	0.076	0.275	0.219	0.174
1	BaggingRegressor	0.897	0.019	0.903	0.903	0.065	0.254	0.203	0.035
2	RandomForestRegressor	0.835	0.025	0.835	0.835	0.11	0.332	0.265	0.746
3	VotingRegressor	0.818	0.127	0.208	0.208	0.53	0.728	0.618	0.165
4	AdaBoostRegressor	0.772	0.032	0.778	0.778	0.149	0.386	0.308	0.115
5	StackingRegressor	0.897	0.019	0.87	0.87	0.087	0.295	0.239	0.431

These models performed better than the baseline models, but still not good enough to achieve the goal of a MSE close to 0 but i was already under a margin of 0.1.

Quick thing to note but very important is looking at the traing time with this size of a dataset, the training time of these models is very low, however i will always favor a simpler model over (LinearReggression) over any ensemble. Why ? If you take a look at the CSV files,the training time of the LinearReggression model is significantly lower than the training time of the ensemble models. In fact the training times are almost 100 times lower than the training time of the ensemble models.

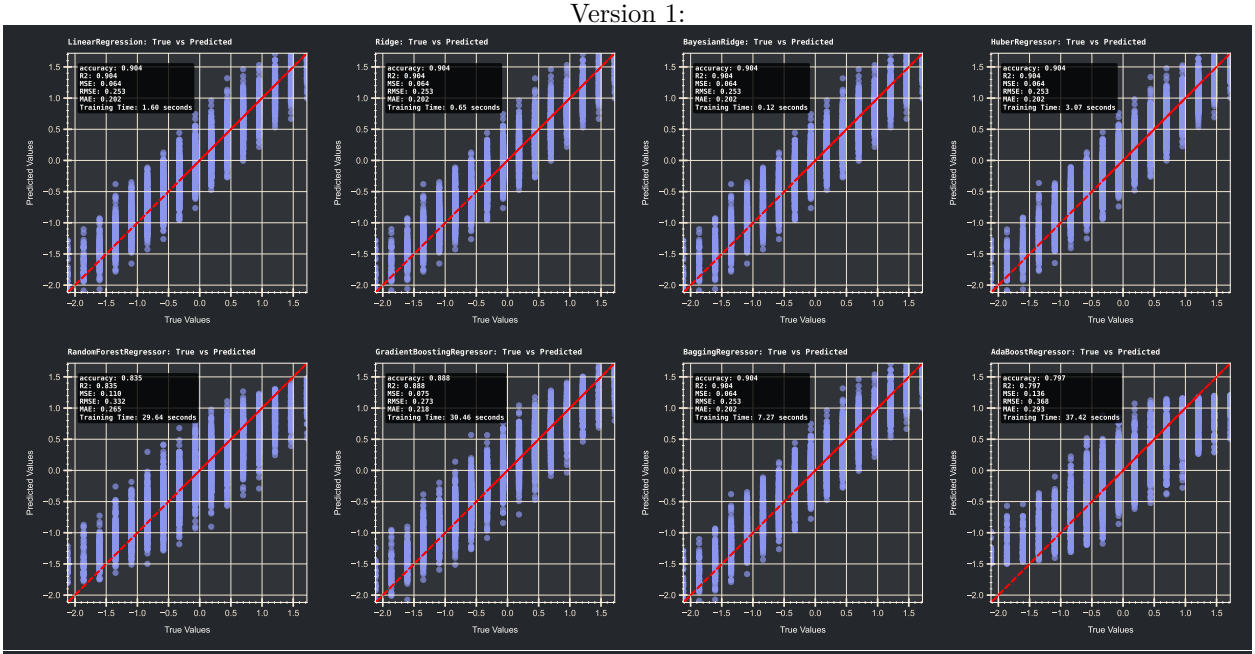
However , we continue our search for the sake of education and to see if we can achieve a MSE close to 0.

2.3 Hyperparameter Tuning

2.3.1 Tuning

The next step was to tune the hyperparameters of the best models to improve their performance. I first used RandomSearchCV to search for the best hyperparameters for all the best models in our previous.

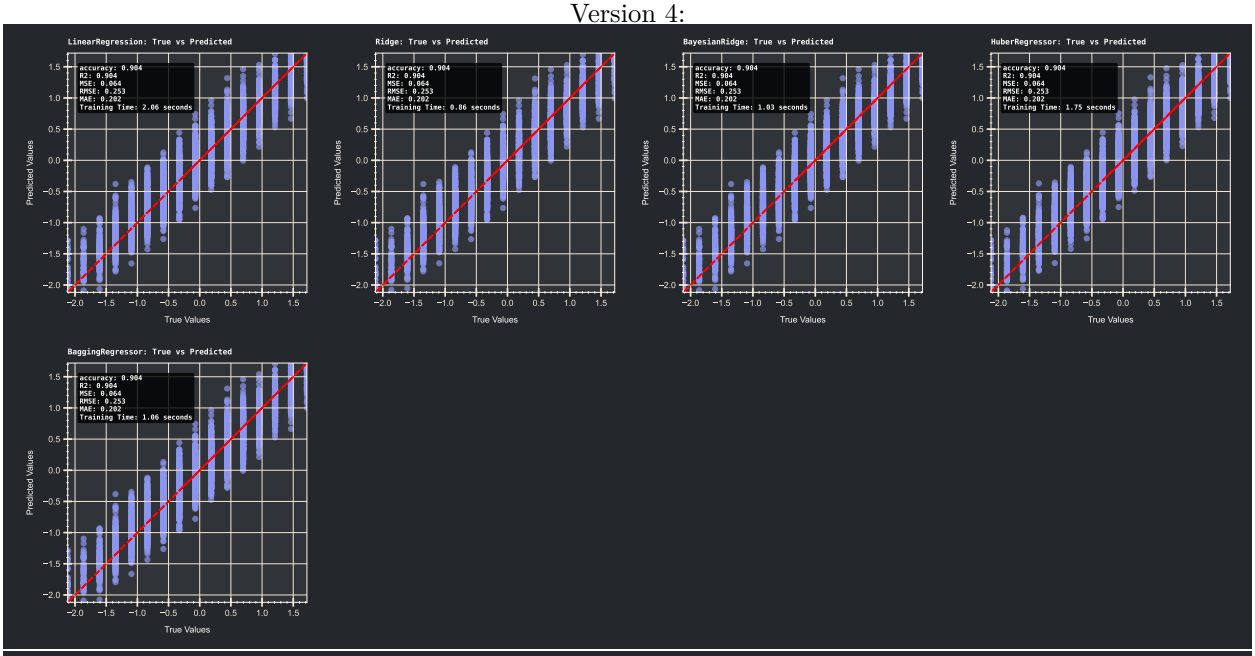
Version 1:



	Model	Cross-Validation		Metrics					Time	Parameters
index	Model	cv_mean_r2	cv_std_r2	accuracy	R2	MSE	RMSE	MAE	training_time	best_params
0	LinearRegression	0.897	0.0	0.904	0.904	0.064	0.253	0.202	1.603	
1	Ridge	0.897	0.0	0.904	0.904	0.064	0.253	0.202	0.649	regressor__alpha: 1.2
2	BayesianRidge	0.897	0.0	0.904	0.904	0.064	0.253	0.202	0.119	regressor__tol: 0.0001 regressor__max_iter: 500 regressor__lambda_2: 1e-08 regressor__lambda_1: 0.0001 regressor__alpha_2: 1e-05 regressor__alpha_1: 0.0001
3	HuberRegressor	0.897	0.0	0.904	0.904	0.064	0.253	0.202	3.074	regressor__tol: 1e-06 regressor__max_iter: 450 regressor__epsilon: 1.6 regressor__alpha: 0.6
4	RandomForestRegressor	0.824	0.007	0.835	0.835	0.11	0.332	0.265	29.642	regressor__n_estimators: 100 regressor__min_samples_split: 2 regressor__min_samples_leaf: 1 regressor__max_depth: 25
5	GradientBoostingRegressor	0.866	0.017	0.888	0.888	0.075	0.273	0.218	30.456	regressor__subsample: 0.6 regressor__n_estimators: 200 regressor__min_samples_split: 10 regressor__min_samples_leaf: 1 regressor__max_depth: 5 regressor__learning_rate: 0.05
6	BaggingRegressor	0.81	0.111	0.904	0.904	0.064	0.253	0.202	7.267	regressor__n_estimators: 100 regressor__max_samples: 0.5 regressor__max_features: 1.0
7	AdaBoostRegressor	0.736	0.038	0.797	0.797	0.136	0.368	0.293	37.422	regressor__n_estimators: 300 regressor__learning_rate: 0.3

And then I used GridSearchCV to search for the best hyperparameters for the best model from the previous step and eliminating the ones that perform less good along the way. repeating this process until I found the best hyperparameters for the best model.

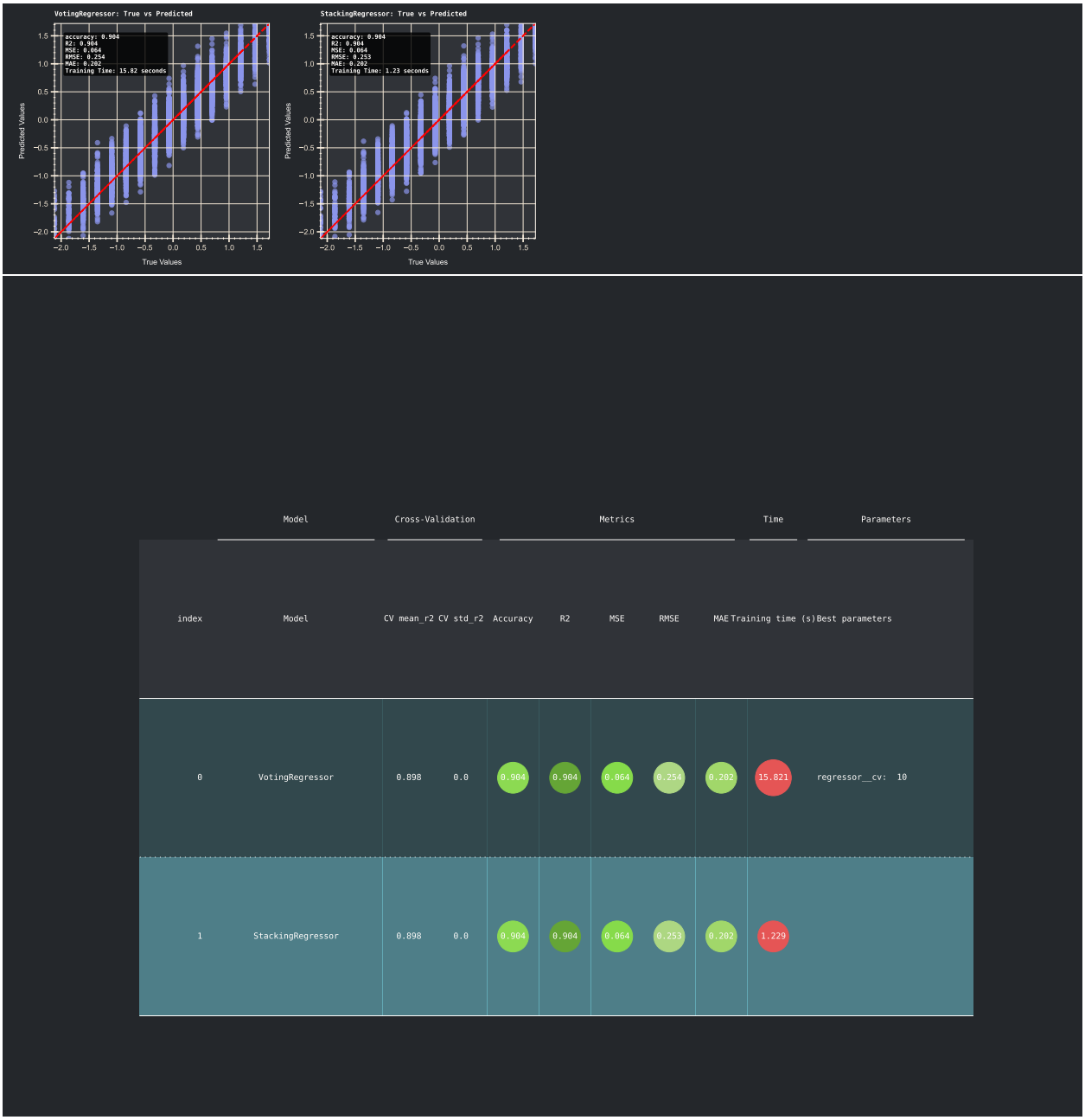
Version 4:



	Model	Cross-Validation		Metrics					Time	Parameters
index	Model	CV mean_r2	CV std_r2	Accuracy	R2	MSE	RMSE	MAE	Training time (s)	Best parameters
0	LinearRegression	0.897	0.0	0.904	0.904	0.064	0.253	0.202	2.064	
1	Ridge	0.897	0.0	0.904	0.904	0.064	0.253	0.202	0.864	regressor__alpha: 3.5
2	BayesianRidge	0.897	0.0	0.904	0.904	0.064	0.253	0.202	1.026	regressor__alpha_1: 1.5e-05 regressor__alpha_2: 1.5e-05 regressor__lambda_1: 0.0005 regressor__lambda_2: 1e-10 regressor__max_iter: 280 regressor__tol: 0.00012
3	HuberRegressor	0.897	0.0	0.904	0.904	0.064	0.253	0.202	1.751	regressor__alpha: 0.7 regressor__epsilon: 1.7 regressor__max_iter: 490 regressor__tol: 1e-07
4	BaggingRegressor	0.897	0.0	0.904	0.904	0.064	0.253	0.202	1.06	regressor__max_features: 1.0 regressor__max_samples: 0.5 regressor__n_estimators: 80

By the time i reached version 7 i had already achieved a MSE of 0.1 and i was not able to improve the performance of the models any further. Using the best models and their best hyperparameters, i was able to achieve a MSE of below 0.1 on the test set. Then loading the best model and using ensemble methods again it will yield the same results to a lack of diversity in the models.

Ensemble 1:

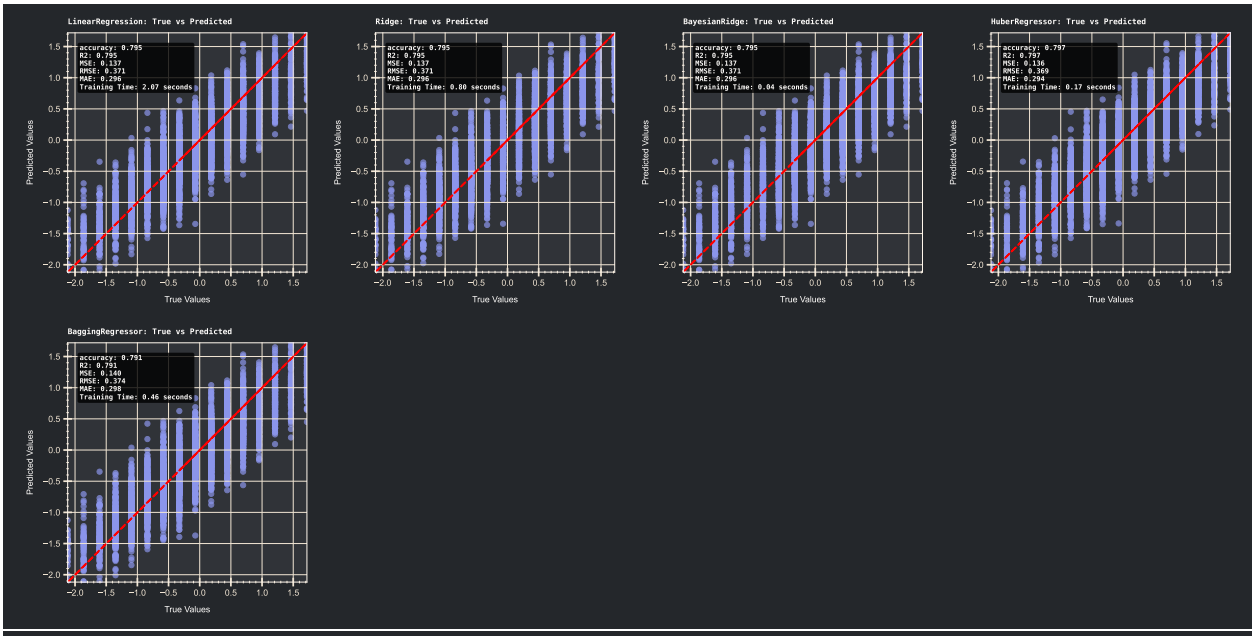


2.3.2 Final Model

The final best models were:

1. **Linear Regression** with a MSE of 0.064 on the test set
2. **Ridge** with a MSE of 0.064 on the test set
3. **Bayesian Ridge** with a MSE of 0.064 on the test set
4. **BaggingRegressor** with a MSE of 0.064 on the test set
5. **HuberRegressor** with a MSE of 0.064 on the test set
6. **VotingRegressor** with a MSE of 0.064 on the test set
7. **StackingRegressor** with a MSE of 0.064 on the test set and with as final estimator a **Linear Regression** model.

I did try to remove the features that i created earlier to see if it would improve the performance of the model, but it did not. In fact the performance of the model was worse.



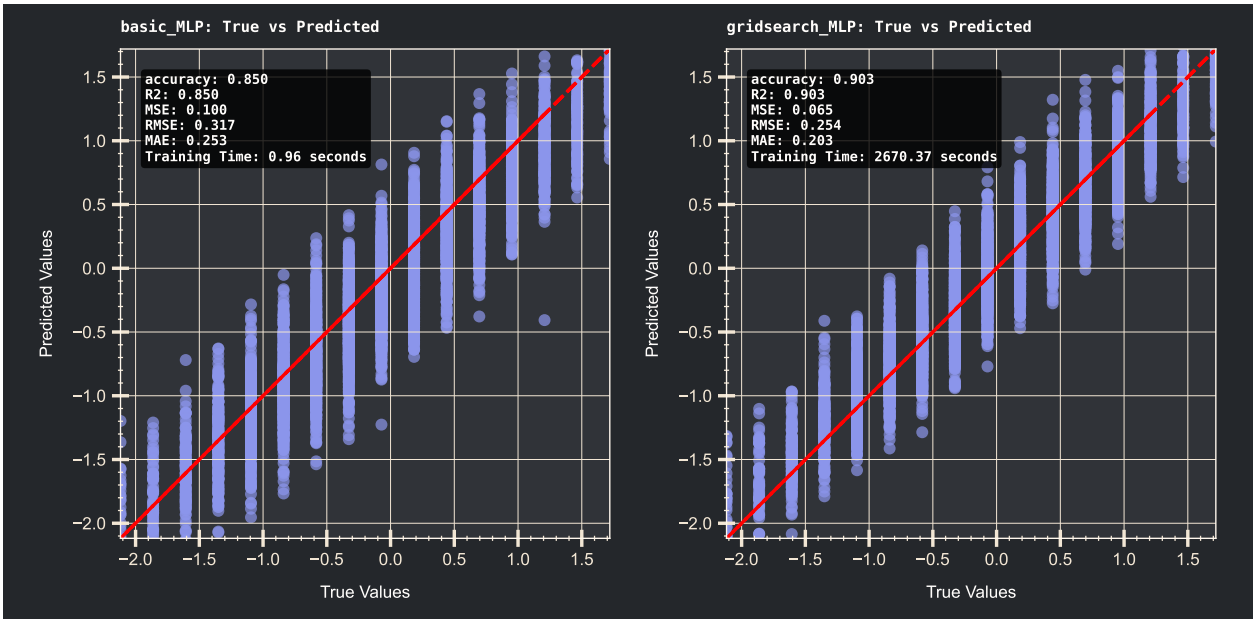
	Model	Cross-Validation		Metrics					Time	Parameters
index	Model	CV mean_r2	CV std_r2	Accuracy	R2	MSE	RMSE	MAE	Training time (s)	Best parameters
0	LinearRegression	0.785	0.0	0.795	0.795	0.137	0.371	0.296	2.067	
1	Ridge	0.786	0.0	0.795	0.795	0.137	0.371	0.296	0.802	regressor__alpha: 3.5
2	BayesianRidge	0.786	0.0	0.795	0.795	0.137	0.371	0.296	0.044	regressor__alpha_1: 1.5e-05 regressor__alpha_2: 1.5e-05 regressor__lambda_1: 1.0 regressor__lambda_2: 1e-10 regressor__max_iter: 1 regressor__tol: 0.00012
3	HuberRegressor	0.785	0.0	0.797	0.797	0.136	0.369	0.294	0.174	regressor__alpha: 0.7 regressor__epsilon: 1.7 regressor__max_iter: 550 regressor__tol: 1e-08
4	BaggingRegressor	0.786	0.0	0.791	0.791	0.14	0.374	0.298	0.455	regressor__max_features: 1.0 regressor__max_samples: 0.5 regressor__n_estimators: 80

I could also have tried to use more advanced models such as XGBoost, LightGBM, CatBoost, etc... but i did not have the time to do so. A other that i could have done is to test different combination of features to see if it would improve the performance of the model.

2.4 Neural Networks

However i did try to use a neural network to see if it would improve the performance of the model. I first tried to use scikit-learn's MLPRegressor, which is a simple feedforward neural network.

```
searchCV = GridSearchCV(
    MLPRegressor(random_state=RANDOM_SEED),
    cv=10,
    param_grid={
        'hidden_layer_sizes': [(32,), (64,), (32, 32), (64, 32), (128, 64)],
        'activation': ['relu', 'tanh'],
        'solver': ['lbfgs'],
        'learning_rate_init': [0.001, 0.01, 0.1],
        'max_iter': [500, 550, 600]
    }
).fit(X_TRAIN,Y_TRAIN)
```



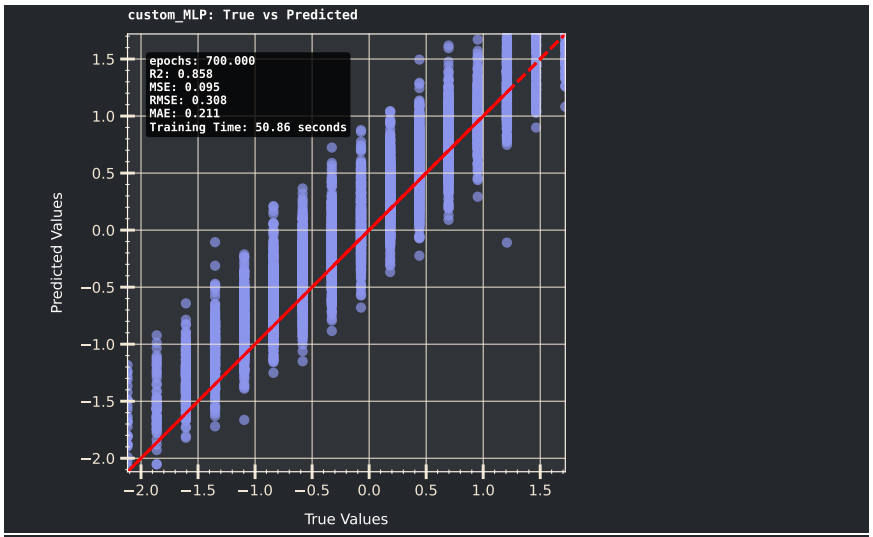
These models did not perform as well as the previous ML models. So i decided to use TensorFlow and Keras to customize the NN further. I had already a pretty good idea on how the NN should like due to the previous step.

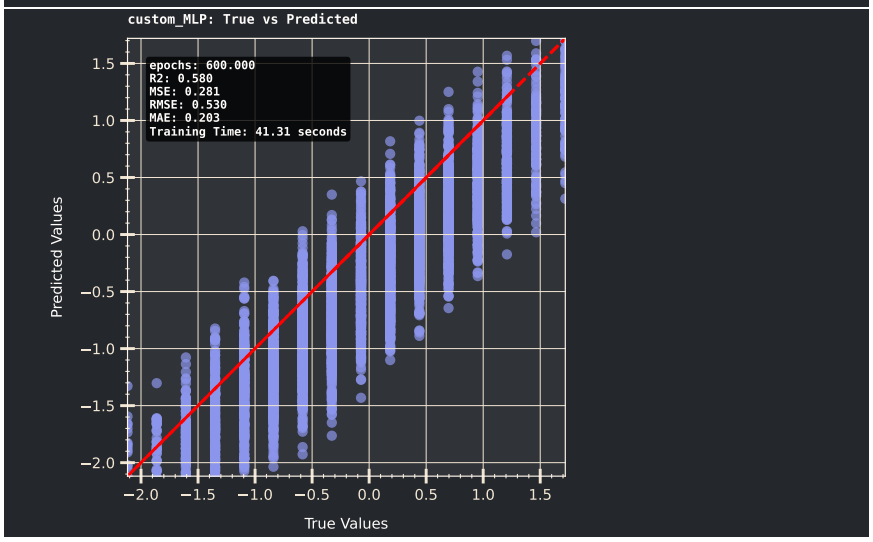
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=X_TRAIN.shape[1:]),
    tf.keras.layers.Dense(1)
])

early_stopping = tf.keras.callbacks.EarlyStopping(
    patience=10,
    restore_best_weights=True,
    verbose=1
)
history: tf.keras.callbacks.History = model.fit(
    X_TRAIN,
    Y_TRAIN,
    epochs=nbr_of_epochs,
    batch_size=ceil(X_TRAIN.shape[0] / 10),
    validation_data=(X_EVAL,Y_EVAL),
    callbacks=[
        early_stopping
    ],
)
```

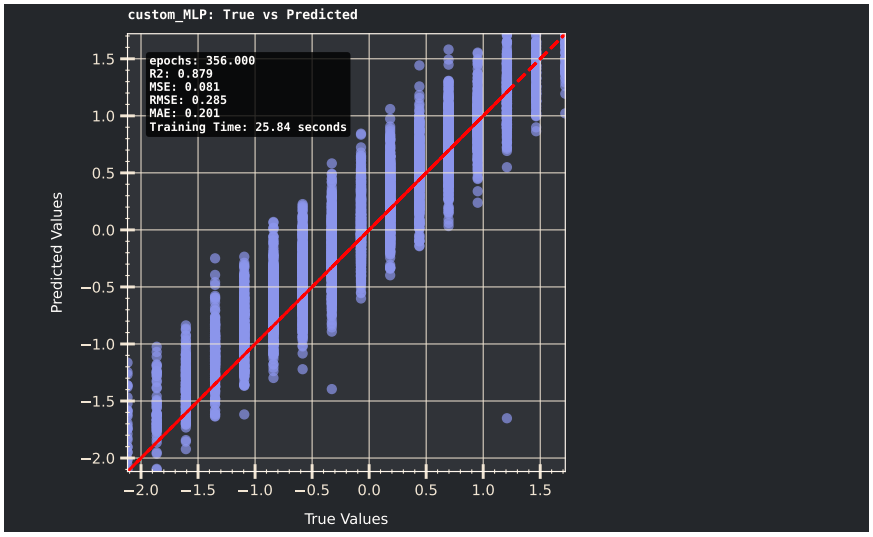
This probably is the most simple NN you can find consisting of 1 input layer , 1 hidden layer , and 1 output layer. The input layer has the same number of neurons as the number of features in the dataset that is given to it, the hidden layer has 32 neurons and the output layer has 1 neuron (because it's a regression task). The first versions of the NN were not that good , sometimes overshooting and sometimes undershooting because i didn't use early stopping callback yet.

No Early Stopping:





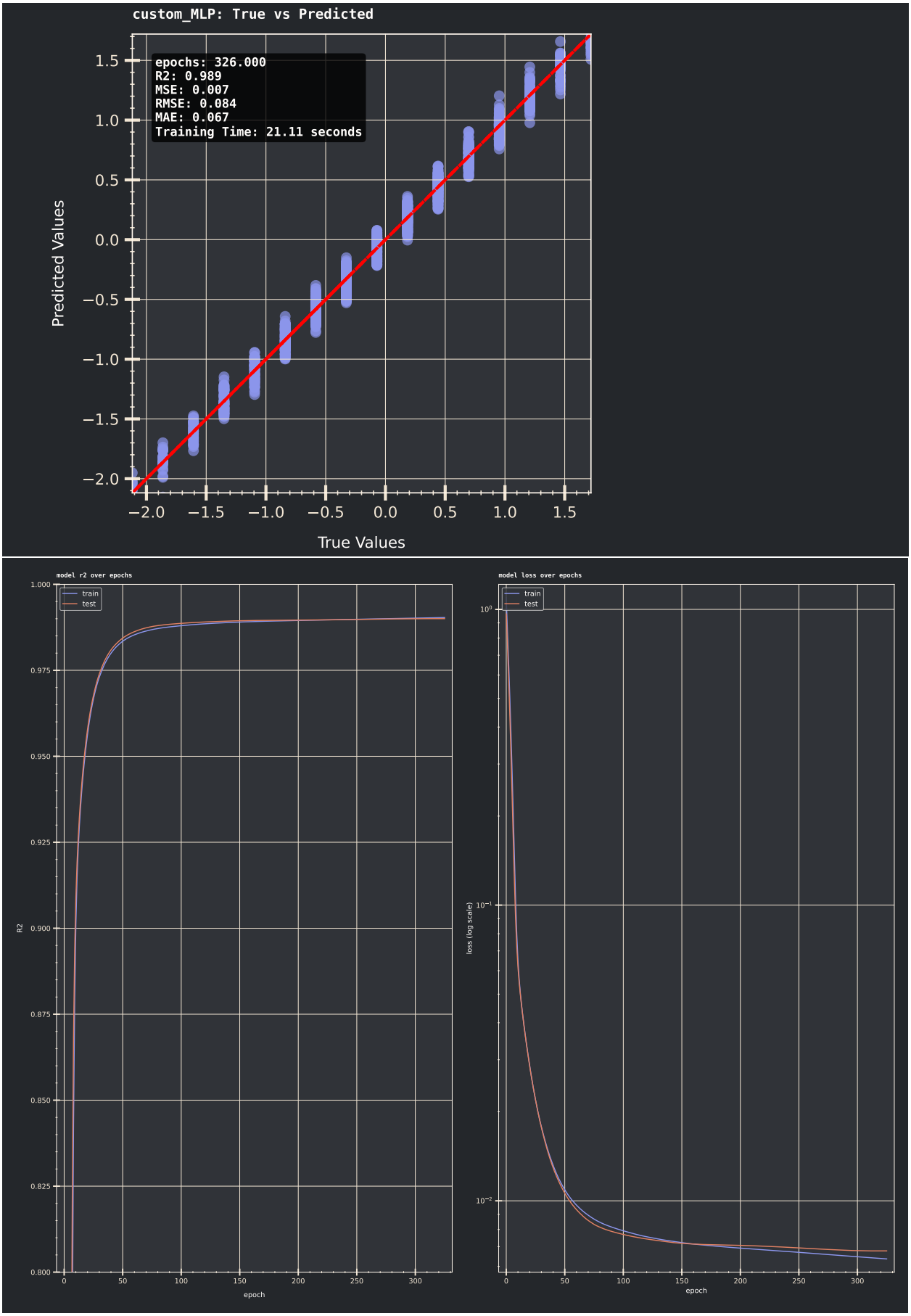
With Early Stopping:



With early stopping the model was able to achieve a MSE of 0.081 on the test set. However the training time was significantly higher than the previous ML models and the MSE is still higher then these previous explored models.

My last attempt was to use the full data set to train the model and see if it would improve the performance of the model.

Full Dataset:



Yeah this is probably the best model i could come up with, having a MSE of 0.00711 on the test set. significantly outperforming the previous ML models and the previous NN models. But again the training time was

significantly higher than the previous ML models. with a ratio of

$$\mathbf{t}_{NN} = 0.00092 \text{ seconds} \tag{0-1}$$

$$\mathbf{t}_{LinearReggression} = 21.10520 \text{ seconds} \tag{0-2}$$

$$\text{Ratio} = \frac{\mathbf{t}_{LinearReggression}}{\mathbf{t}_{NN}} = \frac{21.10520}{0.00092} \approx 22940.43 \tag{0-3}$$

So the training time of the NN is about 22940 times higher than the training time of the Linear Regression model.

3 SHAP

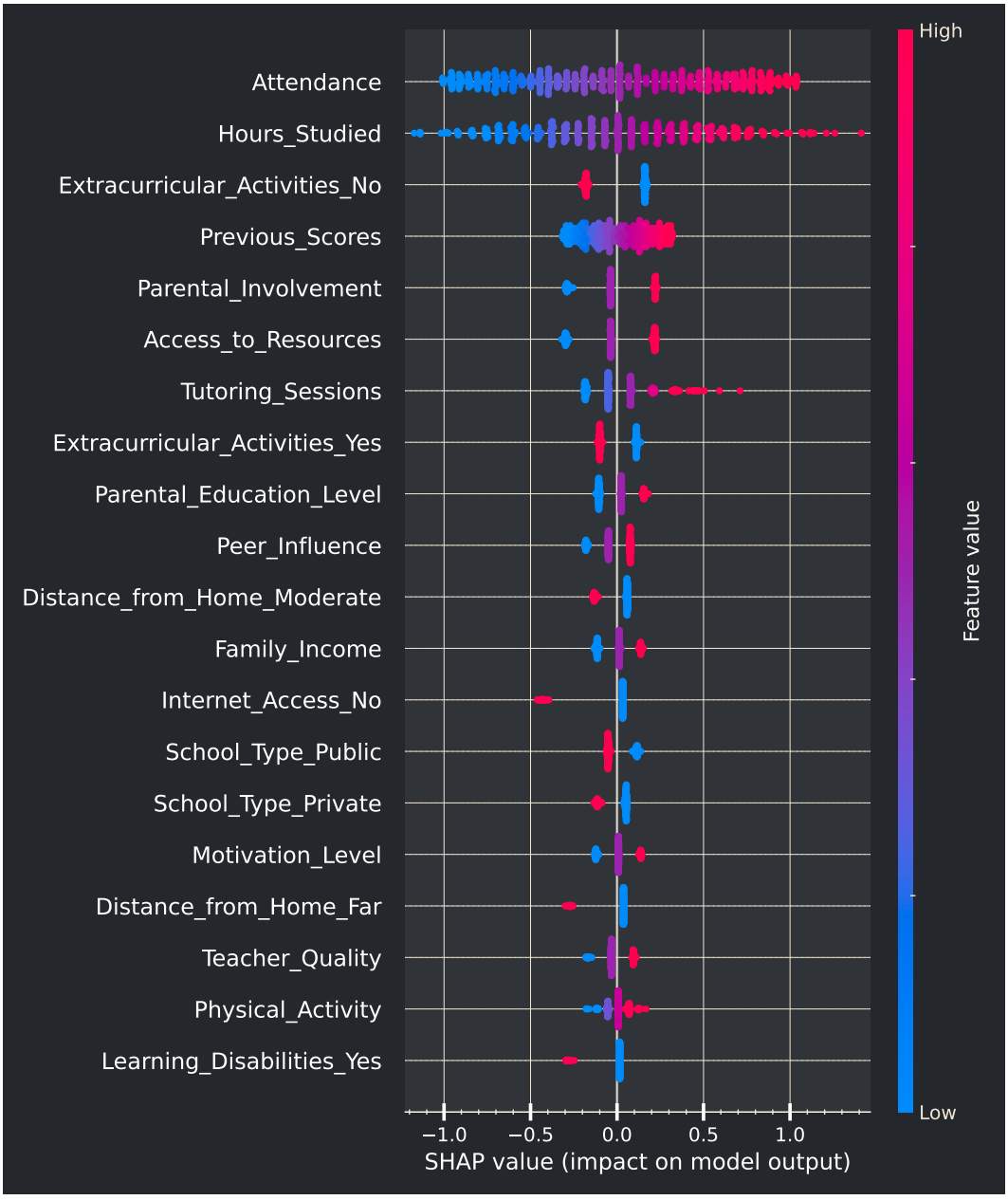
3.1 SHAP Analysis

To understand the impact of each feature on the model's predictions, I used SHAP (SHapley Additive exPlanations) values. SHAP values provide a way to explain the output of any machine learning model by assigning each feature an importance value for a particular prediction.

3.2 SHAP Values Calculation

I used the SHAP library to calculate the SHAP values for the best model (the Neural Network) and then visualized the results using a beeswarm plot.

```
    X_sample = X_TEST.sample(min(sample_size, len(X_TEST)), random_state=RANDOM_SEED)
    explainer = shap.Explainer(model, X_sample)
    shap_values = explainer.shap_values(X_sample)
    shap.summary_plot(shap_values, X_sample,show=False,axis_color="#ffffff")
```



What we can see here is how each feature contributes to the model's predictions. The beeswarm plot shows the SHAP values for each feature, with the color indicating the feature value (red for high values, blue for low values).

4 Conclusion

I could conclude that the best model for this dataset is the Neural Network with a MSE of 0.00711 on the test set. However the training time of the NN is significantly higher than the previous ML models, which makes it less suitable for this dataset. The previous ML models were able to achieve a MSE of 0.064 on the test set, which is still a good result. I could have used the whole dataset on the ML models and used PCA to reduce the dimensionality of the dataset, but i did not have the time to do so. Nor did i have the time to try different combination of features to see if it would improve the performance of the model. In conclusion, the project was a success in terms of achieving the goal of building a machine learning model that can predict the students exam score based on the various factors provided in the dataset.

Submitted by Celik Ennis on May 31, 2025.