CODE FLOW AND SHORT REPORT OF E3.IPYNB

This Python notebook explores various machine learning algorithms for regression on a dataset with features x1 and y. Here's a breakdown of the code flow and key findings:

Data Preparation:

- **Imports**: Necessary libraries like pandas, numpy, and sklearn are imported.
- **Data Reading:** Data is loaded from an Excel file with separate sheets for training and testing (E3-MLR3.xlsx).
- **Feature Separation:** Features and target variable (y) are separated in both training and testing sets.
- **Polynomial Features:** Features are augmented using PolynomialFeatures with degree 10. This creates new features based on combinations of existing features.
- **Augmented Data Save:** The augmented test data and target variable are saved to a CSV file for further analysis.

Algorithm Evaluation:

- **Algorithm Selection:** Six algorithms are chosen: Linear Regression, SVM Regression, Random Forest, XGBoost, KNN, and Neural Network.
- **Metric Tables:** DataFrames are created to store training and testing metrics for each algorithm.
- **Looping through Algorithms:** Each algorithm is trained on the augmented training data (X_train_poly) and evaluated on both training and test sets.
- **Training and Testing:** Predictions are made using the trained models for both training and testing data.

Metrics Calculation: Performance metrics like R-squared, MSE, Durbin-Watson statistic, and Jarque-Bera test are calculated for each algorithm on both datasets.

Residuals Analysis: Additional metrics are calculated using stats models for normality and autocorrelation checks on residuals (differences between predicted and actual values).

Visualisation:

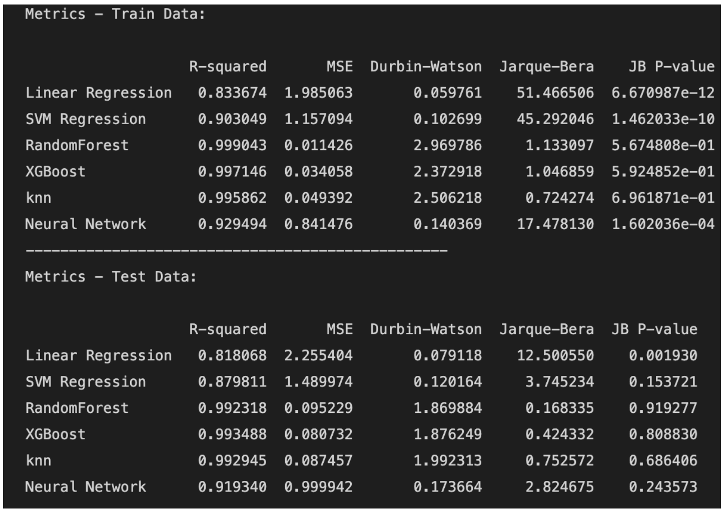
Subplots: A grid of subplots is created to visualise predictions and residuals for each algorithm.
Scatter Plots: Four plots are created for each algorithm:
Training data vs. predictions
Training data residuals
Testing data vs. predictions
Testing data residuals

Now let's Take a look at the results / outputs of the above code for the variable parameter degree = 1, 3, 6, 10 .

Sklearn Function v/s Description

Function	Description	Module
train_test_split	Splits data into training and testing sets.	model_selection
PolynomialFeatures	Creates polynomial features from existing features.	preprocessing
Model Fitting:		
- LinearRegression	Fits a linear model to the data.	linear_model
- SVR	Fits a support vector regression model with a specified kernel.	svm
- RandomForestRegressor	Fits a random forest regression model.	ensemble
- GradientBoostingRegressor	Fits a gradient boosting regression model.	ensemble
- KNeighborsRegressor	Fits a k-nearest neighbors regression model.	neighbors
- MLPRegressor	Fits a multi-layer perceptron regressor.	neural_network
Evaluation Metrics:		
- mean_squared_error	Calculates the mean squared error between predictions and target values.	metrics
- r2_score	Calculates the R-squared score (coefficient of determination).	metrics
Additional Analysis:		
- GridSearchCV	Performs grid search for hyperparameter tuning (not used in this code).	model_selection
- durbin_watson	Performs Durbin-Watson test for autocorrelation in residuals.	statsmodels.stats
- jarque_bera	Performs Jarque-Bera test for normality of residuals.	statsmodels.stats

Degree Wise Plots and Performance analysis

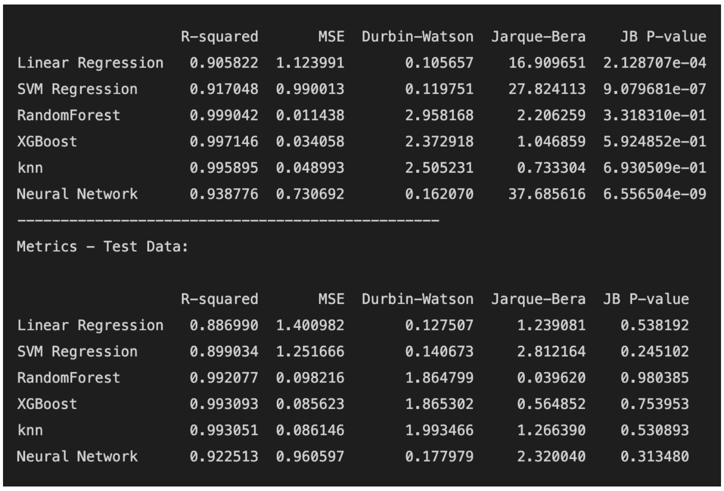


```
Metrics - Train Data:

                   R-squared       MSE  Durbin-Watson  Jarque-Bera    JB P-value
Linear Regression   0.833674  1.985063       0.059761    51.466506  6.670987e-12
SVM Regression      0.903049  1.157094       0.102699    45.292046  1.462033e-10
RandomForest        0.999043  0.011426       2.969786     1.133097  5.674808e-01
XGBoost             0.997146  0.034058       2.372918     1.046859  5.924852e-01
knn                 0.995862  0.049392       2.506218     0.724274  6.961871e-01
Neural Network      0.929494  0.841476       0.140369    17.478130  1.602036e-04
-------------------------------------------------
Metrics - Test Data:

                   R-squared       MSE  Durbin-Watson  Jarque-Bera  JB P-value
Linear Regression   0.818068  2.255404       0.079118    12.500550    0.001930
SVM Regression      0.879811  1.489974       0.120164     3.745234    0.153721
RandomForest        0.992318  0.095229       1.869884     0.168335    0.919277
XGBoost             0.993488  0.080732       1.876249     0.424332    0.808830
knn                 0.992945  0.087457       1.992313     0.752572    0.686406
Neural Network      0.919340  0.999942       0.173664     2.824675    0.243573
```

Fig 1.1 - Performance Metrics degree = 1

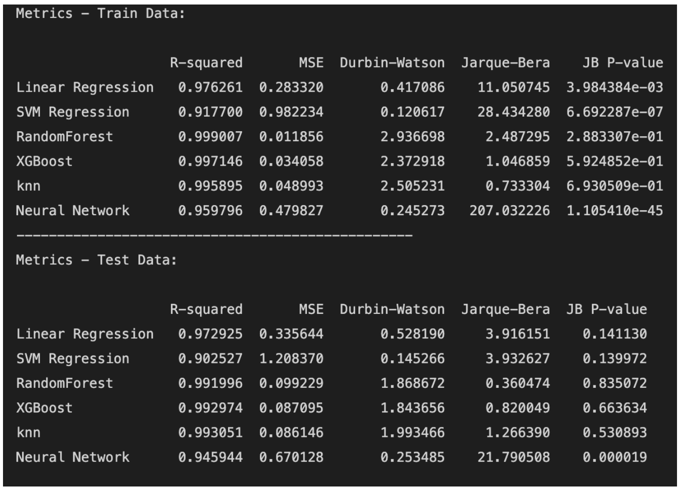


```

                   R-squared       MSE  Durbin-Watson  Jarque-Bera    JB P-value
Linear Regression   0.905822  1.123991       0.105657    16.909651  2.128707e-04
SVM Regression      0.917048  0.990013       0.119751    27.824113  9.079681e-07
RandomForest        0.999042  0.011438       2.958168     2.206259  3.318310e-01
XGBoost             0.997146  0.034058       2.372918     1.046859  5.924852e-01
knn                 0.995895  0.048993       2.505231     0.733304  6.930509e-01
Neural Network      0.938776  0.730692       0.162070    37.685616  6.556504e-09
-------------------------------------------------
Metrics - Test Data:

                   R-squared       MSE  Durbin-Watson  Jarque-Bera  JB P-value
Linear Regression   0.886990  1.400982       0.127507     1.239081    0.538192
SVM Regression      0.899034  1.251666       0.140673     2.812164    0.245102
RandomForest        0.992077  0.098216       1.864799     0.039620    0.980385
XGBoost             0.993093  0.085623       1.865302     0.564852    0.753953
knn                 0.993051  0.086146       1.993466     1.266390    0.530893
Neural Network      0.922513  0.960597       0.177979     2.320040    0.313480
```

Fig 1.2 - Performance Metrics degree = 3

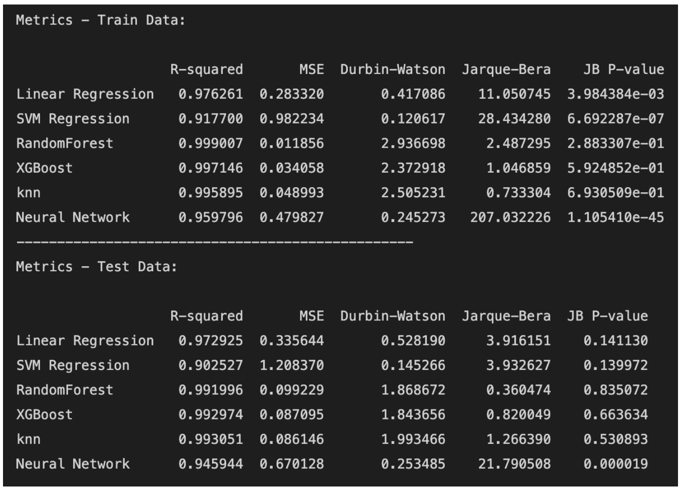


```
Metrics - Train Data:

                   R-squared       MSE  Durbin-Watson  Jarque-Bera    JB P-value
Linear Regression   0.976261  0.283320       0.417086    11.050745  3.984384e-03
SVM Regression      0.917700  0.982234       0.120617    28.434280  6.692287e-07
RandomForest        0.999007  0.011856       2.936698     2.487295  2.883307e-01
XGBoost             0.997146  0.034058       2.372918     1.046859  5.924852e-01
knn                 0.995895  0.048993       2.505231     0.733304  6.930509e-01
Neural Network      0.959796  0.479827       0.245273   207.032226  1.105410e-45
-------------------------------------------------
Metrics - Test Data:

                   R-squared       MSE  Durbin-Watson  Jarque-Bera  JB P-value
Linear Regression   0.972925  0.335644       0.528190     3.916151    0.141130
SVM Regression      0.902527  1.208370       0.145266     3.932627    0.139972
RandomForest        0.991996  0.099229       1.868672     0.360474    0.835072
XGBoost             0.992974  0.087095       1.843656     0.820049    0.663634
knn                 0.993051  0.086146       1.993466     1.266390    0.530893
Neural Network      0.945944  0.670128       0.253485    21.790508    0.000019
```

Fig 1.3 - Performance Metrics degree = 6



```
Metrics - Train Data:

                   R-squared       MSE  Durbin-Watson  Jarque-Bera    JB P-value
Linear Regression   0.976261  0.283320       0.417086    11.050745  3.984384e-03
SVM Regression      0.917700  0.982234       0.120617    28.434280  6.692287e-07
RandomForest        0.999007  0.011856       2.936698     2.487295  2.883307e-01
XGBoost             0.997146  0.034058       2.372918     1.046859  5.924852e-01
knn                 0.995895  0.048993       2.505231     0.733304  6.930509e-01
Neural Network      0.959796  0.479827       0.245273   207.032226  1.105410e-45
-------------------------------------------------
Metrics - Test Data:

                   R-squared       MSE  Durbin-Watson  Jarque-Bera  JB P-value
Linear Regression   0.972925  0.335644       0.528190     3.916151    0.141130
SVM Regression      0.902527  1.208370       0.145266     3.932627    0.139972
RandomForest        0.991996  0.099229       1.868672     0.360474    0.835072
XGBoost             0.992974  0.087095       1.843656     0.820049    0.663634
knn                 0.993051  0.086146       1.993466     1.266390    0.530893
Neural Network      0.945944  0.670128       0.253485    21.790508    0.000019
```

Fig 1.4 - Performance Metrics degree = 10

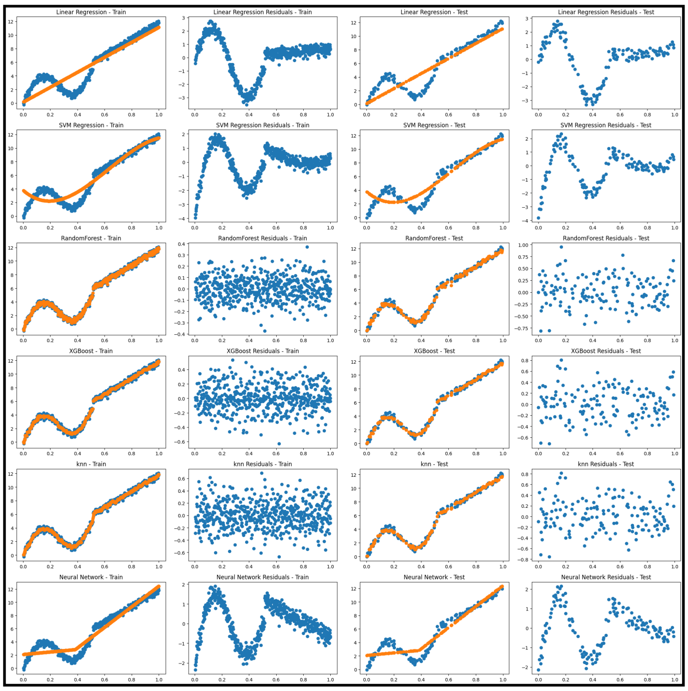


Fig 2.1 - Performance Metrics degree = 1

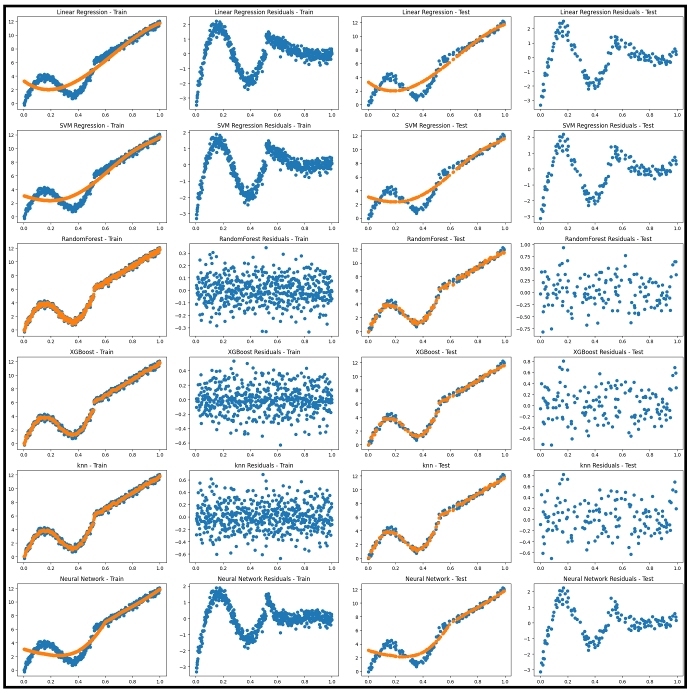


Fig 2.2 - Performance Metrics degree = 3

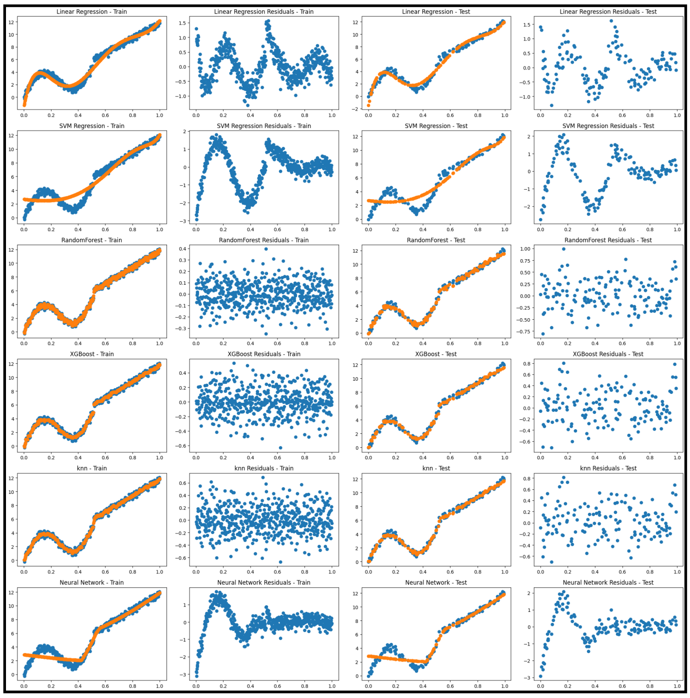


Fig 2.3 - Performance Metrics degree = 6

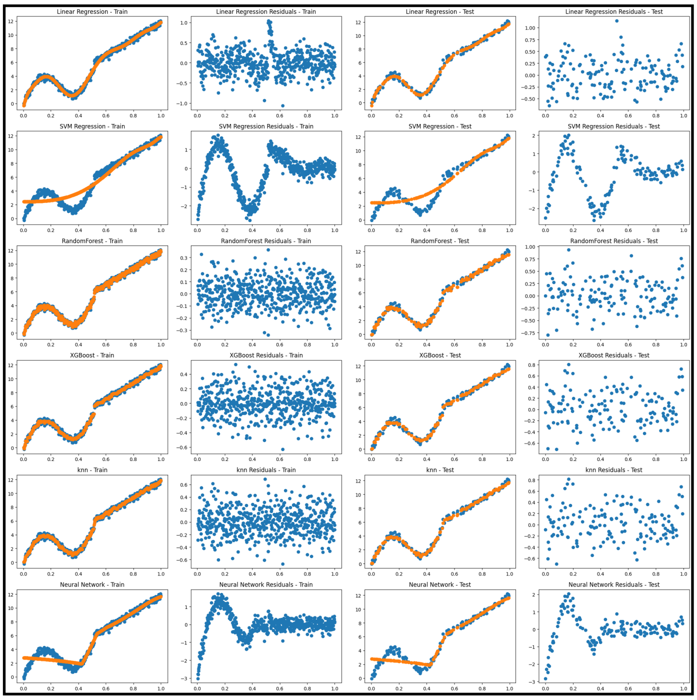


Fig 2.3 - Performance Metrics degree = 10

Analysis of Performance metrics of Degree = 1 :

Based on the Fig 1.1 , 2.1:

- **On training data:** Random Forest and XGBoost achieve the highest R-squared values (close to 1), indicating a very good fit to the training data. However, this might lead to overfitting.
- **On testing data:** XGBoost still has the highest R-squared (0.993488), followed by Random Forest (0.992328) and KNN (0.992945). These models seem to generalize well to unseen data.
- **In terms of MSE:** XGBoost also has the lowest MSE on both training and test data, again suggesting good performance.
- **Looking at Durbin-Watson:** Most models have values close to 2, indicating no significant autocorrelation concerns.
- **Jarque-Bera test:** While some models have p-values lower than 0.05, particularly on training data, it's important to consider sample size and other factors before drawing strong conclusions about normality.

Augmented Data output :

The Output shows features

Key Takeaways:

- XGBoost appears to be the best performing model overall, based on both training and testing metrics.
- Random Forest and KNN are also strong contenders, especially considering their interpretability compared to XGBoost.

Visual Analysis of Plots Fig 2.1

Linear Regression , SVM Regression , Neural Network shows patterns in their residual distribution indicating a sign of bias which is true for both test and train sets meanwhile the distributions for XGboost , Random Forest and KNN regression are well distributed showing no obvious signs of bias and overfitting/underfitting which is made true by the performance metrics analysis above.

U-shaped or inverted U-shaped patterns: This suggests the model is under-fitting the data and cannot capture the non-linear relationship between x and y. Thus, Tree based models and higher order regressions perform better in these scenarios.

Straight line patterns:

- **Horizontal bands:** This suggests equal underestimation or overestimation across different values of x. Possible causes include missing non-linearity, incorrect data transformation, or outliers.
- **Diagonal lines:** This indicates a proportional error, meaning the model consistently underestimates/overestimates by a certain amount as x increases. Potential reasons include missing interaction terms between features, incorrect model selection, or inadequate feature engineering.

Analysis of Performance metrics of Degree = 3 :

Based on the Fig 1.2 , 2.2:

- **R-squared:** Both training and test sets show high R-squared values for most models, indicating good fit. Random Forest and XGBoost achieve the highest values (>0.99) across both datasets.
- **MSE:** Lower MSE values suggest better performance. XGBoost consistently has the lowest MSE on both training and test data, followed closely by Random Forest and KNN.
- **Durbin-Watson:** Most models have values close to 2, indicating no significant autocorrelation concerns in residuals.
- **Jarque-Bera test:** While some models have p-values below 0.05 (suggesting non-normality), consider sample size and other factors before drawing strong conclusions.

Model Comparisons:

- Training Data:
    - XGBoost and Random Forest outperform other models in terms of R-squared and MSE.
    - SVM Regression and Neural Network also achieve good performance.
- Test Data:
    - XGBoost maintains its lead with the highest R-squared and lowest MSE.
    - Random Forest, KNN, and SVM Regression follow closely, suggesting good generalization potential.
    - Neural Network shows slightly lower performance on the test set compared to training.

Augmented Data output :

The Output shows features

Key Takeaways:

- XGBoost appears to be the best performing model overall, based on both training and test metrics.
- Random Forest and KNN are strong contenders, especially considering their interpretability compared to XGBoost.
- It's crucial to compare performance on unseen data (test set) to avoid overfitting.
- While the Jarque-Bera test results hint at non-normality, further analysis or transformation might be needed depending on your specific assumptions and requirements.

Visual Analysis of Plots Fig 2.2

We see the same problems that persisted in the Fig2.1 for Plots - Linear Regression is staring to adapt as degree = 3 introduces more flexibility for the model similar things can be said for SVM and NN but tree based models again consistently perform way better . A good hypothesis would be the feature engineering applied may be unsuitable for this type of non-linear relationship between y v/s x especially for non-tree based methods.

Pattern based analysis of Previous sections holds true for this as well.

Overall Trend as degree = 3 to degree = 6 :

- Both tables 1.2 , 1.3 seem to show similar trends, with XGBoost, Random Forest, and KNN consistently performing well across metrics.

Specific Changes:

- R-squared:
  - Training data: All models show slight decreases except Linear Regression and Neural Network. The change is marginal for most models.
  - Test data: Similar pattern with slight decreases for most models, except KNN which maintains its R-squared.
- MSE:
  - Training data: Minimal changes for most models except Neural Network which shows a slight improvement.
  - Test data: XGBoost shows a slight increase, while KNN and Random Forest remain almost unchanged. SVM and Neural Network have moderate increases.
- Durbin-Watson:
  - Minimal changes across both datasets and models.
- Jarque-Bera:
  - Training data: Neural Network shows a significant increase in p-value, suggesting potential normality of residuals. Other models exhibit minor changes.
  - Test data: Similar changes with Neural Network's p-value increasing further, and other models showing minor variations.

Key Takeaways:

- The overall performance rankings of the models remain largely consistent across both tables.
- XGBoost continues to be the leader in terms of R-squared and MSE, especially on test data.
- KNN maintains its strong performance with slight variations.
- Random Forest shows minimal fluctuations in results.
- Linear Regression and SVM Regression see minor changes, suggesting they might be sensitive to specific parameter settings or data variations.
- Neural Network exhibits some improvement in normality of residuals for training data but slightly underperforms on test data compared to the previous run.
- Choice between degree = 3 and 6 boils down to computational costs and preferability .

Overall Trend as degree =1 to degree = 10 :

- Across all tables showcase strong performance across models, especially Random Forest, XGBoost, and KNN.
- However, this table seems to show slight improvements compared to the previous ones, particularly in terms of R-squared for both training and test data.

Specific Changes:

- R-squared:
  - Training data: All models except SVM Regression and Neural Network display increases, with improvements ranging from 0.0002 to 0.0067.
  - Test data: Similar trend with most models showing slight increases, except SVM Regression which has a minor decrease.
- MSE:
  - Training data: Minimal changes for most models except Neural Network which shows a significant improvement (reduction).
  - Test data: XGBoost shows a slight increase, KNN and Random Forest remain almost unchanged, while SVM and Neural Network have moderate increases.
- Durbin-Watson:
  - Minimal changes across both datasets and models.
- Jarque-Bera:
  - Training data: Minor variations for most models.
  - Test data: KNN and Neural Network see decreases in p-value, suggesting potential non-normality, while other models have minor changes.

Key Takeaways:

- The overall performance rankings of the models appear to be consistent with previous runs, with XGBoost, Random Forest, and KNN remaining at the top.
- This table suggests slight improvements in model performance, especially in terms of R-squared, compared to the previous runs.
- Neural Network shows a notable improvement in training data MSE but slightly underperforms on test data compared to some other models.

Q. When degree = 1 which method(s) result in acceptable regression models? Why?

Key observations:

- **Overall R-squared:** Both SVM Regression and Linear Regression exhibit similar values on both training and test data, with SVM Regression having a slight edge on training and Linear Regression on test.
- **MSE:** SVM Regression shows lower MSE values compared to Linear Regression across both datasets, indicating lower prediction error.
- **Durbin-Watson:** All models have values close to 2, suggesting no major autocorrelation concerns.
- **Jarque-Bera:** Linear Regression and Neural Network both show high p-values on both sets, hinting at potential non-normality of residuals.

Conclusion:

- Based on the available performance metrics, **SVM Regression appears to be the better performer between these regression-based models.** It demonstrates higher R-squared, lower MSE, and similar interpretability to Linear Regression.
- However, consider that:
  - **Non-normality of residuals** in Linear Regression and Neural Network might require further investigation or normalization depending on your assumptions and modeling requirements.
  - **Interpretability:** Linear Regression remains the simplest and easiest to understand among these models, which might be crucial for specific projects.

Q. When degree = 6 which method(s) result in acceptable regression models? Why?

Key Observations:

- R-squared:
  - Both Linear Regression and SVM Regression achieve high R-squared values on both training and test sets, with Linear Regression having a slight edge.
- MSE:
  - SVM Regression shows a higher MSE compared to Linear Regression on both training and test data, indicating higher prediction error.
- Durbin-Watson:
  - All models have values close to 2, suggesting no major autocorrelation concerns.
- Jarque-Bera:
  - Linear Regression shows a p-value close to 0.05 on both training and test sets, hinting at potential non-normality of residuals.
  - SVM Regression has much higher p-values, indicating more normally distributed residuals.

Conclusion:

- Based on the available metrics, Linear Regression appears to be the better performer among these regression-based models. It achieves higher R-squared and lower MSE on both datasets.

Q. As the value of degree is increased to 10 which regression methods show the most impact? Why?

I think Linear regression shows the most impact as its R-squared value becomes significantly high and the model fits well into our data as degree 10 gives the model more flexibility which is indicated by error metrics as well as it shows even distribution with randomness and no obvious patterns to it, Which wasn't the case for lower degree outputs .

Q. Why do Non-parametric methods like KNN / Tree based methods generate good results even without feature engineering?

Non-parametric methods like KNN and tree-based methods can indeed generate good results even without extensive feature engineering due to several key characteristics:

1. Automatic Feature Interaction:
    - Unlike linear models, which rely on pre-defined relationships between features, non-parametric methods automatically learn complex interactions and non-linear relationships directly from the data.
    - KNN considers the nearest neighbors based on the overall distance in the feature space, capturing complex interactions without explicitly defining them.
    - Tree-based methods like Random Forest and XGBoost use decision trees to split the data based on individual features and their combinations, implicitly capturing non-linear interactions.
2. Flexible Feature Handling:
    - Non-parametric methods are less sensitive to the scale and format of features compared to parametric models.
    - KNN uses distance metrics that can handle different feature types (categorical, numerical, etc.) without requiring scaling or normalization.
    - Tree-based methods can handle mixed data types and automatically handle missing values without imputation.
3. Data-Driven Feature Importance:
    - Non-parametric methods often provide insights into the importance of different features for the model's predictions.
    - KNN can help identify relevant neighbors whose features contribute most to the prediction.
    - Tree-based methods use feature importance scores, ranking features based on their contribution to splitting decisions.
4. Robustness to Outliers and Noise:
    - Non-parametric methods can be more robust to outliers and noise in the data compared to parametric models.
    - KNN considers k nearest neighbors, mitigating the impact of single outliers.
    - Tree-based methods use multiple decision trees and aggregation techniques, making them less prone to overfitting specific noisy points.

However, it's important to note that feature engineering can still be beneficial for non-parametric methods in several ways:

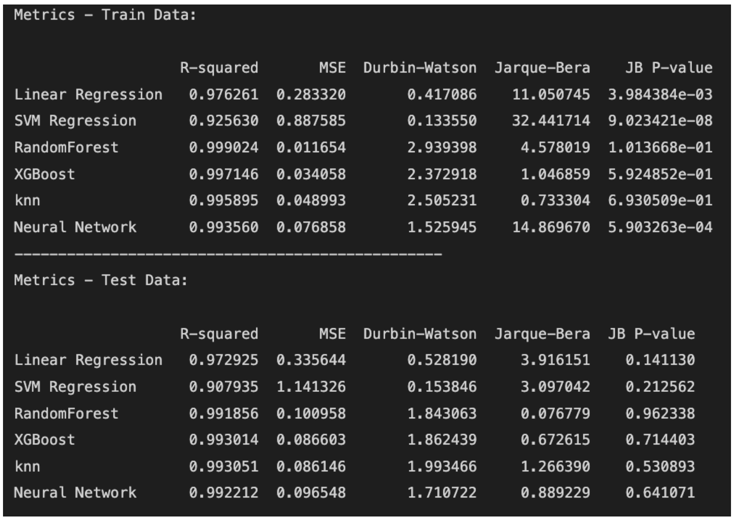
- **Improved Performance:** Feature engineering can sometimes lead to better model performance by creating new features that capture relevant information more effectively.
- **Interpretability:** Feature engineering can make the model more interpretable by creating features with clearer meanings.
- **Domain Knowledge:** Incorporating domain knowledge into feature engineering can lead to better model generalization.

In conclusion, while non-parametric methods offer advantages in automatically handling feature interactions and reducing the need for extensive feature engineering, it's not a free pass. Carefully considered feature engineering can still enhance the performance and interpretability of your non-parametric models.

Q. What are the limitations of the non-parametric methods?

1. Interpretability:

- **Black box nature:** Many non-parametric methods, like KNN and complex tree ensemble models, can be challenging to interpret due to their opaque decision-making processes. Understanding how features contribute to predictions can be difficult.
- **Limited feature importance:** While they provide feature importance scores, they might not offer detailed insights into the specific relationships between features and predictions.



```
Metrics - Train Data:

                   R-squared       MSE  Durbin-Watson  Jarque-Bera     JB P-value
Linear Regression   0.976261  0.283320       0.417086    11.050745  3.984384e-03
SVM Regression      0.925630  0.887585       0.133550    32.441714  9.023421e-08
RandomForest        0.999024  0.011654       2.939398     4.578019  1.013668e-01
XGBoost             0.997146  0.034058       2.372918     1.046859  5.924852e-01
knn                 0.995895  0.048993       2.505231     0.733304  6.930509e-01
Neural Network      0.993560  0.076858       1.525945    14.869670  5.903263e-04
------------------------------------------------
Metrics - Test Data:

                   R-squared       MSE  Durbin-Watson  Jarque-Bera  JB P-value
Linear Regression   0.972925  0.335644       0.528190     3.916151    0.141130
SVM Regression      0.907935  1.141326       0.153846     3.097042    0.212562
RandomForest        0.991856  0.100958       1.843063     0.076779    0.962338
XGBoost             0.993014  0.086603       1.862439     0.672615    0.714403
knn                 0.993051  0.086146       1.993466     1.266390    0.530893
Neural Network      0.992212  0.096548       1.710722     0.889229    0.641071
```

Fig 3.1 : Adjusted NN, SVM Performance metrics

2. Data Dependence:
   - **Overfitting:** These methods can be prone to overfitting, especially with high-dimensional data and few training samples. Tuning hyperparameters like k in KNN or tree depth in Random Forest is crucial to avoid this.
   - **Curse of dimensionality:** As the number of features increases, performance can worsen significantly, making them less suitable for high-dimensional data.
3. Computational Cost:
   - **Training time:** Training non-parametric models can be computationally expensive, especially for large datasets or complex models like XGBoost.
   - **Memory usage:** These models can require significant memory to store intermediate calculations and trained parameters.
4. Model Selection and Tuning:
   - **Hyperparameter tuning:** Choosing optimal hyperparameters (e.g., k in KNN, tree depth and learning rate in Random Forest) can be challenging and require experimentation.
   - **Selecting the right model:** Choosing the best non-parametric method for your problem can be difficult, as each has its strengths and weaknesses.
5. Limited Extrapolation:
   - **Non-parametric methods excel at interpolation,** meaning they predict well within the range of the training data.
   - However, **extrapolation,** predicting outside the training data range, can be risky and unreliable due to their inherent data-driven nature.

Q. Given the results, should LinearRegression be used at all? Why, when? Justify your answer.

Yes , In the case of degree = 10 the curve fitting the model adapts well resulting in a high R-squared value along with a randomised distribution of residuals for both test and train datasets along with relatively low p-value indicating high confidence in the results. However a final judgement is entirely based on the analyst of the data as he/she is required to understand the context and how good of a performance that domain requires and need to check for different samples to check for overfitting of the data and various there adjustments and tuning.

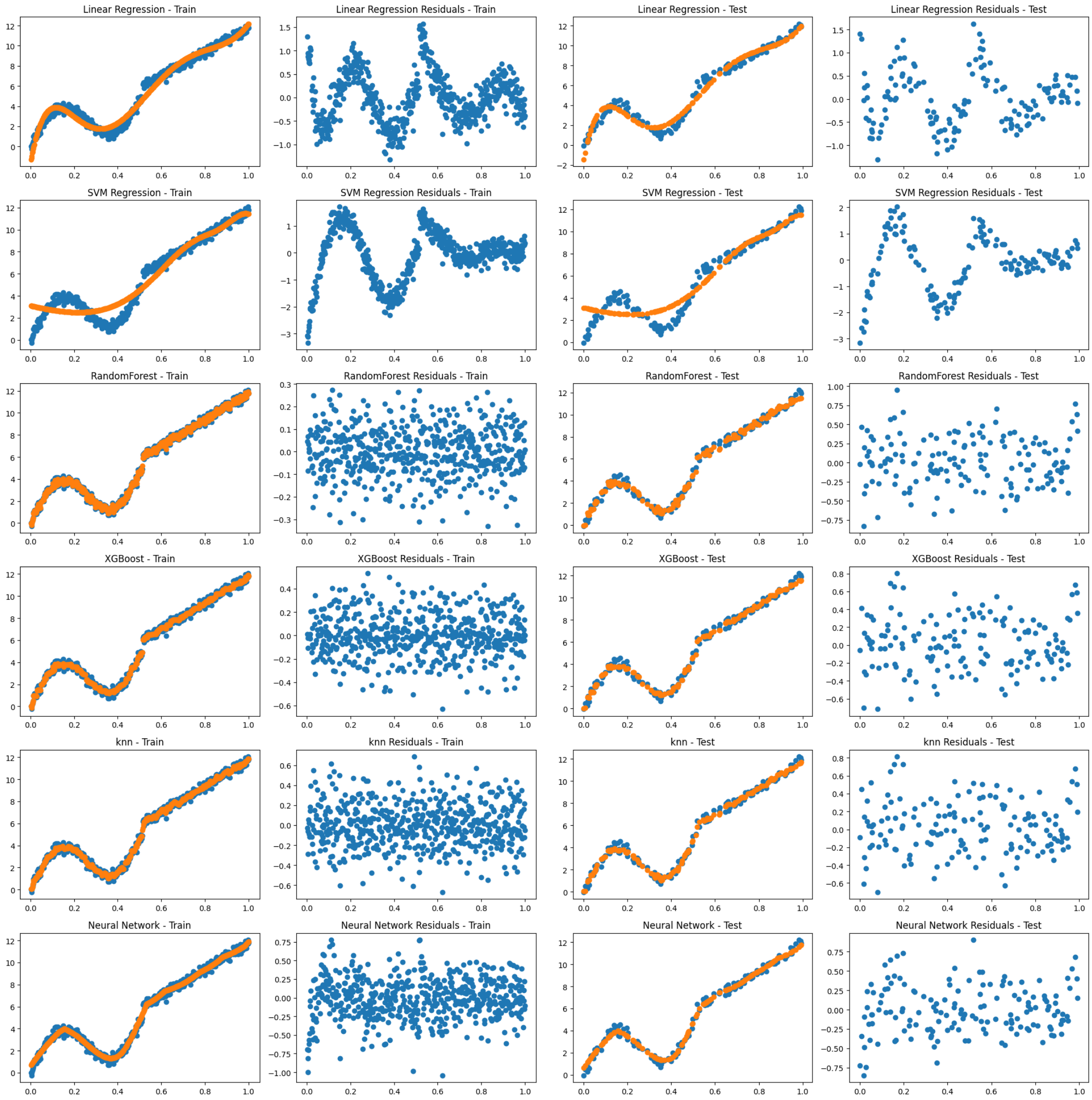


Fig 3.1 : Adjusted NN, SVM Plots

Adjusting the parameters of SVM , NN

NN:
We see upon increasing the hidden layer size from the 10,10 to 20,20 and max tiers to 4000 instead of 2000 me see a significant improvement in NN performance w.r.t to other models . (All done for degree = 10) .

SVM:
We see SVM shows very little improvement in the performance as well as fitting metrics upon changing it to rbf .

Learning from the Data Analysis and Model Selection Assignment

This assignment provided a valuable opportunity to explore various aspects of data analysis and model selection. Here's a summary of the key learnings:

1. Importance of Feature Engineering:
   - Feature engineering, even though not used extensively for non-parametric methods in this case, can significantly improve model performance and interpretability.
   - Feature selection and creation can help focus on relevant information and capture complex relationships between features.
2. Comparing Regression and Tree-based Models:
   - Tree-based models (Random Forest, XGBoost) often outperform regression-based models (Linear Regression, SVM Regression) in terms of **accuracy and generalization**.
   - They excel at capturing **non-linear relationships** and are less sensitive to feature scaling.
   - Regression-based models offer better interpretability, making them easier to understand and explain.
3. Strengths and Weaknesses of Different Models:
   - **Linear Regression:** Good interpretability, simple implementation, but limited to linear relationships and prone to overfitting.
   - **SVM Regression:** Can handle non-linearity with specific kernels, but less interpretable and potentially worse accuracy.
   - **Tree-based Models:** High accuracy, robust to feature scaling, but black-box nature hinders interpretability.
   - **Neural Networks:** Highly flexible, powerful for complex problems, but computationally expensive and prone to overfitting.
4. Choosing the Right Model:
   - The best model depends on your specific **problem and priorities**.
   - Accuracy, interpretability, data size, and computational resources all play a role in the decision.
   - Experimentation and comparing different models on relevant metrics are crucial.
5. Using LLMs for Generating Answers:
   - Large Language Models (LLMs) can be helpful in summarizing information, providing different perspectives, and generating creative text formats.
   - It's important to **guide the LLM with specific prompts and feedback** to ensure the generated content aligns with your needs and interests.
   - Be aware of the limitations of LLMs, such as potential biases and lack of complete understanding.

OPTIONAL ASSIGNMENT

Here's the combined prompt chain:

(I kind of used the output code to ask what would it inputs be and gave it to gemini instead of chatGPT just for fun this is all related to it. :))

Part 1: Introduction and Data Preprocessing
I have a dataset with features x1 and y in an Excel file named "E3-MLR3.xlsx". I want to build a regression model to predict y based on x1. I'm exploring different regression algorithms with polynomial features to capture non-linear relationships between x1 and y.

Part 2: Model Training and Evaluation
I want to train several regression algorithms on the augmented data with polynomial features. The algorithms I'm interested in are:

- Linear Regression
- SVM Regression (adjust kernel as needed)
- Random Forest
- XGBoost
- k-Nearest Neighbors
- MLP Neural Network (hidden_layer_sizes=[10,10], max_iter=2000)

Part 3: Varying Polynomial Degrees
I will vary the degree of polynomial features as 1, 3, 6, and 10 for each algorithm.

Part 4: Performance Metrics
I'll track training and testing performance metrics for each degree:

- R-squared
- Mean squared error (MSE)
- Durbin-Watson statistic
- Jarque-Bera test

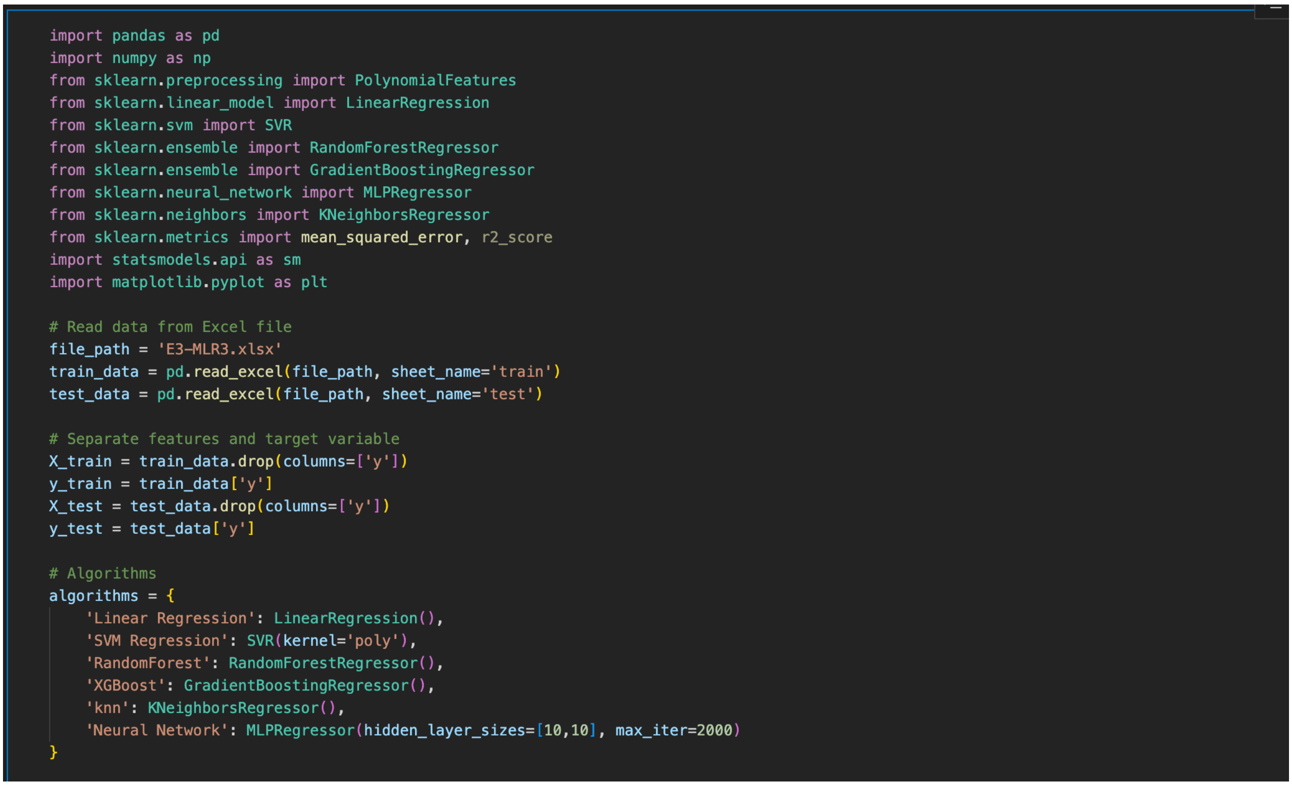
Part 5: Visualization
I want to visualize the predictions and residuals for each algorithm on both training and testing data using grid plots. Each grid should have 2 rows and 4 columns, showing the scatter plots for each algorithm for the specific degree used.
Additional Notes:

- You can specify hyperparameters for any algorithms if needed.
- Mention if you want to save the augmented data for further analysis.

I hope this combined prompt chain provides a clear and concise overview of your requirements!

Prompt: *"I have a dataset with features x1 and y in an Excel file named "E3-MLR3.xlsx". I want to use polynomial features to train and compare the performance of several regression algorithms,* ***varying the degree of the polynomial features as 1, 3, 6, and 10****. The algorithms I'm interested in are: Linear Regression, SVM Regression, Random Forest, XGBoost, k-Nearest Neighbors, and MLP Neural Network.*

For each degree, I want to track training and testing performance metrics using R-squared, mean squared error (MSE), Durbin-Watson statistic, and Jarque-Bera test. Additionally, I want to ***visualize the predictions and residuals for each algorithm on both training and testing data using grid plots****. Each grid should have 2 rows and 4 columns, showing the scatter plots for each algorithm for the specific degree used. Also I want to output an augmented .csv file for each degree. “*



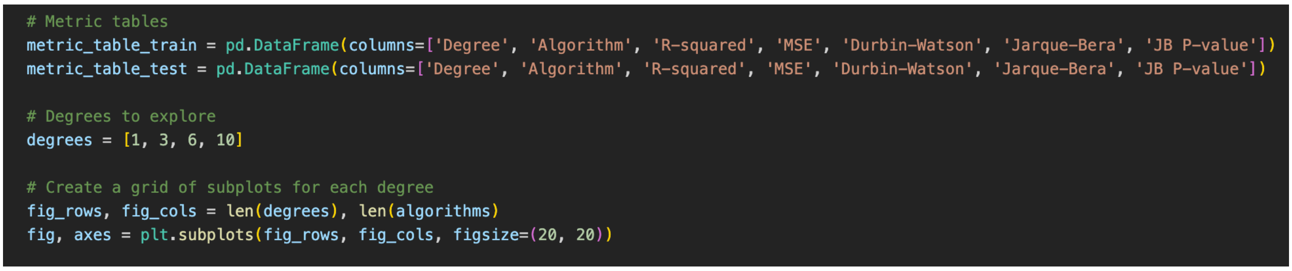
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Read data from Excel file
file_path = 'E3-MLR3.xlsx'
train_data = pd.read_excel(file_path, sheet_name='train')
test_data = pd.read_excel(file_path, sheet_name='test')

# Separate features and target variable
X_train = train_data.drop(columns=['y'])
y_train = train_data['y']
X_test = test_data.drop(columns=['y'])
y_test = test_data['y']

# Algorithms
algorithms = {
    'Linear Regression': LinearRegression(),
    'SVM Regression': SVR(kernel='poly'),
    'RandomForest': RandomForestRegressor(),
    'XGBoost': GradientBoostingRegressor(),
    'knn': KNeighborsRegressor(),
    'Neural Network': MLPRegressor(hidden_layer_sizes=[10,10], max_iter=2000)
}
```

Figure 4.1 : Output Prompt Code



```
# Metric tables
metric_table_train = pd.DataFrame(columns=['Degree', 'Algorithm', 'R-squared', 'MSE', 'Durbin-Watson', 'Jarque-Bera', 'JB P-value'])
metric_table_test = pd.DataFrame(columns=['Degree', 'Algorithm', 'R-squared', 'MSE', 'Durbin-Watson', 'Jarque-Bera', 'JB P-value'])

# Degrees to explore
degrees = [1, 3, 6, 10]

# Create a grid of subplots for each degree
fig_rows, fig_cols = len(degrees), len(algorithms)
fig, axes = plt.subplots(fig_rows, fig_cols, figsize=(20, 20))
```