Aerodynamic Drag Prediction and Comparative Analysis of Models

Project Proposal for course ME 228 – 2024

Deadline: March 5, 5:00 PM
Stage 1
(5 Marks out of 100 of the course)

Team ID: 10
Member 1: Priyanshi
Member 2: Rohan Mekala
Member 3: Varad Patil

Background

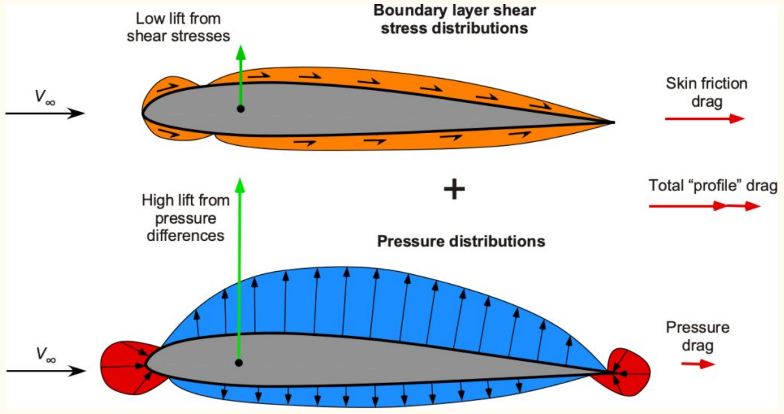
Priyanshi
22B2213

- Where did you find it ?
  - We explored the aerospace industry for problems with available datasets to apply predictive algorithms and optimize flight parameters.
  - A popular topic in this field is drag estimation and optimization for aircraft.
- Why is it important ?
  - Estimating and reducing drag is a topic of great interest in the field of aerodynamics as it helps in improving fuel efficiency, reducing vibrational stresses on the aircraft and improving overall flight performance.
  - We are developing a model to correlate various geometry parameters such as angle of attack, coefficient of lift, etc. with the drag to identify which features affect drag the most and in turn suggest optimizations to reduce the drag.
- How is it solved currently?
  - Various wind tunnel tests and computational fluid dynamics simulations are being conducted in an effort to map the various parameters such as wingspan, tail position and control surface sizing to the drag of the aircraft.
  - Currently, various Gaussian progression as well as Convolution Neural Network models are being implemented to predict aerodynamic coefficients and perform aerodynamic shape optimization to reduce the drag.

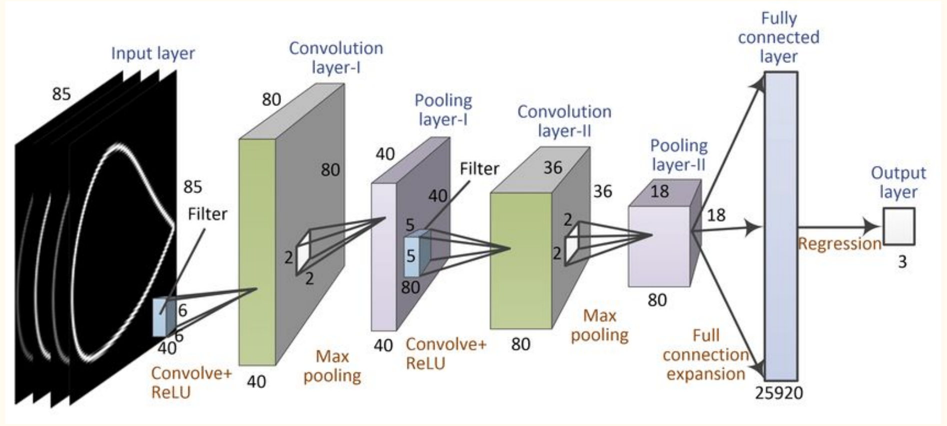
Priyanshi
22B2213

- What are the shortcomings in the current solution ?
  - Due to overfitting data to the model, the model becomes highly biased towards the training data set and displays high variance when tried to fit on any other data sets.
  - Upon conducting the Jarque-Bera test, we have identified that the errors do not follow a normal distribution which further complexifies the solution.
  - We have identified a high Condition Number that suggests the model might be fiting noise or capturing spurious relationships due to its sensitivity to slight data changes. This makes the estimated coefficients less reliable and conclusions less trustworthy.

- Have you been able to find data ?
  - Yes, we have found a data set on kaggle.


Low lift from shear stresses
Boundary layer shear stress distributions
Skin friction drag
V_∞
+
Total "profile" drag
High lift from pressure differences
Pressure distributions
Pressure drag
V_∞

Stress due to drag



CNN model for airfoils

Motivation and Objective

Rohan Mekala
22B2106

Motivation :

- This project is of great importance as it serves as a means of integrating various predictive ML algorithms to real world problems.
- We have chosen the problem of drag estimation and optimization as it resonates the most in terms of the impact that it has, be it on extending the range of flight, increasing the payload capacity or extending the engine life of the aircraft.
- To prevent overfitting of the model, we plan to use adaptive boosting methods such Gradient boosting to allow for the optimization of arbitrarily differentiable loss functions by sequentially executing the decision trees.

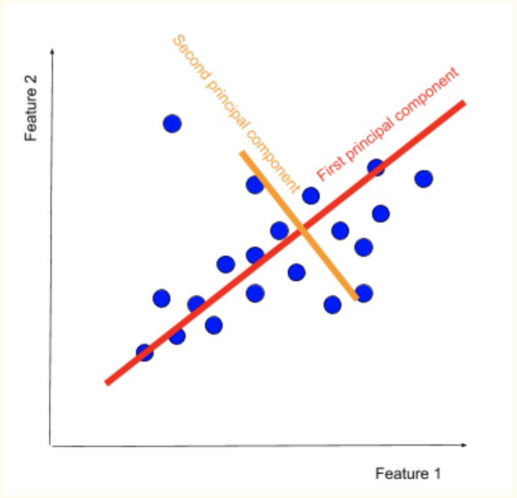
Objective :

- We aim to predict the **drag coefficient** by utilizing a dataset that includes various features including geometrical and fluid properties.
- We then plan to reduce the drag generated by identifying the important features that correlate the most to the drag using various dimensionality reduction techniques such as PCA, thereby giving us the optimal values of parameters such as reynolds number and angle of attack required for minimized drag for different airfoils.
- Additionally, we can also perform a comparative analysis to suggest which airfoil would perform better under different cruise velocities.

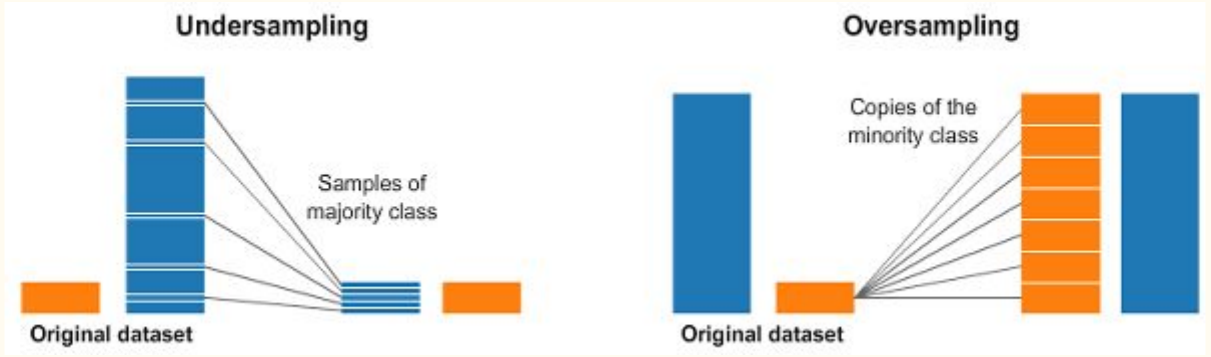
Rohan Mekala
22B2106

Methodology:

- Using imputation techniques for preprocessing data to account for missing values
- Perform under/over sampling for data imbalance
- Performing exploratory data analysis to gauge the initial distribution of the data
- Scaling the data using normalization and standardization along with performing PCA for dimensionality reduction
- Performing train test split using sklearn library
- Validating the model and accordingly adjusting the hyperparameters
- Fitting the regression model on the test data



Principal Component Analysis

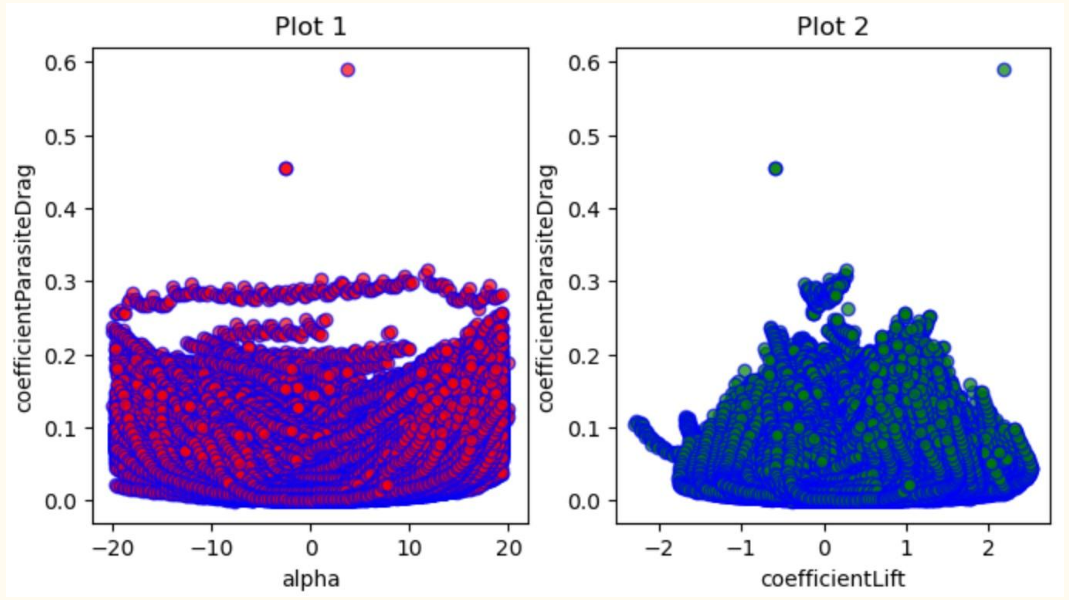


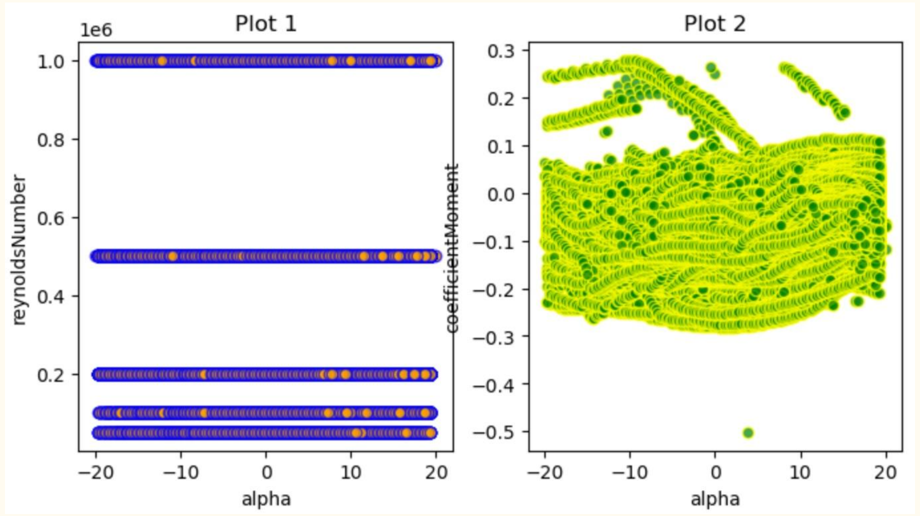
Undersampling and Oversampling

Varad Patil
22B2270

Validation method:

- Performing training-validation splits using sklearn
- Using grid search, we will obtain the optimal configurations of the hyperparameters.
- Performing k-fold cross validation to cluster the observations in k groups and performing validation across various groups as this method is significantly computationally efficient as compared to Leave One Out Cross Validation(LOOCV).
- Validating the results predicted using CFD tools such as Ansys Fluent as well as other Aircraft Performance simulation software such as XFLR5.





Sample distribution of initial data

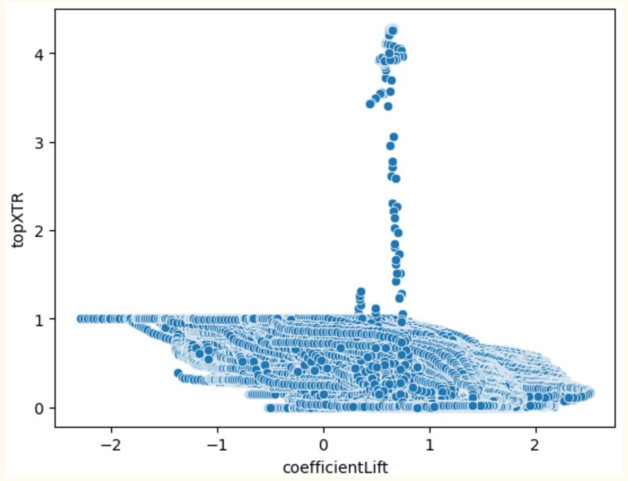
Varad Patil
22B2270

Challenges :-

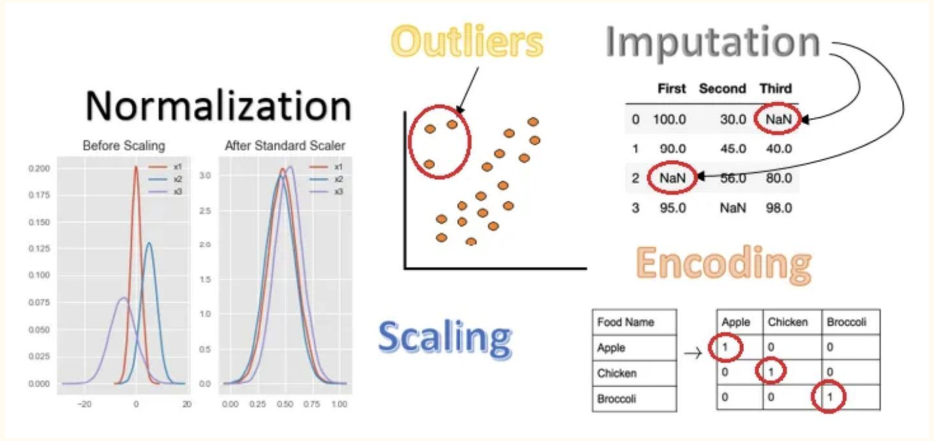
- Data curing was a challenge as some data values faced errors in their conversions to numeric format.
- Overfitting can be a problem as we have a high dimensional feature space (70 features).
- Complex models may be harder to interpret which could prove challenging when our main objective is to perform a comparative study of different models.
- Choosing an inappropriate sampling strategy for training models with big data can drastically affect the model performance and generalization.

Exit Plan :-

- A wrongly configured hyperparameter could lead to a highly unstable model, in the event of which, we would proceed to perform grid search to rectify the issue.
- Any outliers in the data can wrongly skew the predictions of the model. The solution to this would be to either perform winsorization(replacing the outliers with the nearest "good" data point) or to cap the data set to eliminate the outliers.
- Different regression models react differently to scaling leading to errors in some of the models. To avoid this, we would use the appropriate scaling methods for the respective models.



Outliers



	First	Second	Third
0	100.0	30.0	NaN
1	90.0	45.0	40.0
2	NaN	56.0	80.0
3	95.0	NaN	98.0

Food Name
Apple
Chicken
Broccoli

Apple	Chicken	Broccoli
1	0	0
0	1	0
0	0	1

Data Preprocessing

Thank you

Aerodynamic Drag Prediction and Comparative Analysis of Models

Course Project
ME 228 – 2024

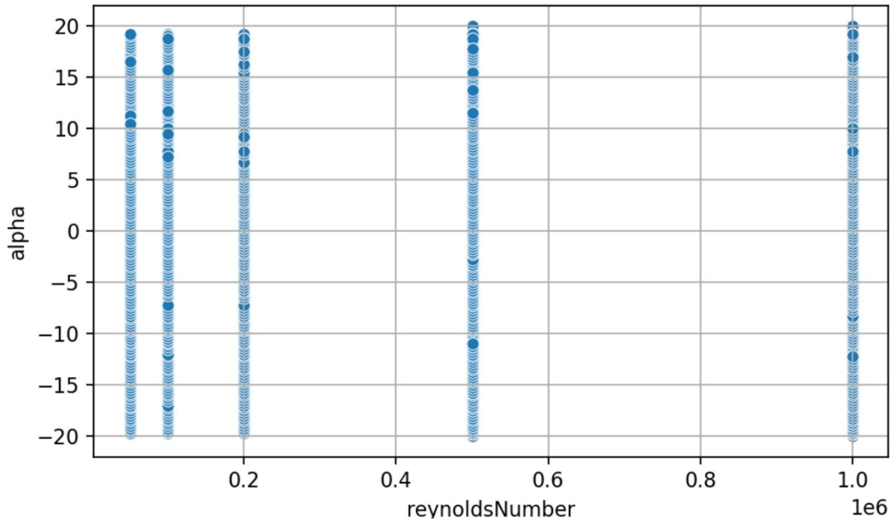
Deadline: April 7, 5:00 PM
Stage 2

Team ID: 10
Member 1: Priyanshi
Member 2: Rohan Mekala
Member 3: Varad Patil

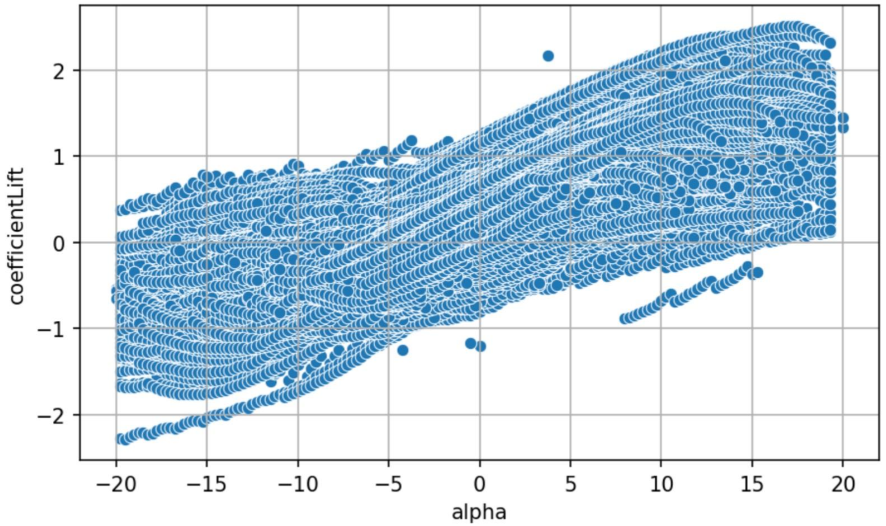
Objectives

- Performing feature analysis using methods such as PCA and SVD and using the identified principal features to estimate the drag on the airfoils.
- Implementing various models such as random forest regression, gradient boosting and Support Vector Machines and performing a comparative analysis to verify which model gives us the estimated drag coefficient(Cd) of the highest accuracy.
- Using various models to evaluate the different parameters such as alpha and coefficient of lift for which the coefficient of drag will be minimum for a particular airfoil at a particular reynolds number.

Priyanshi , 22b2213

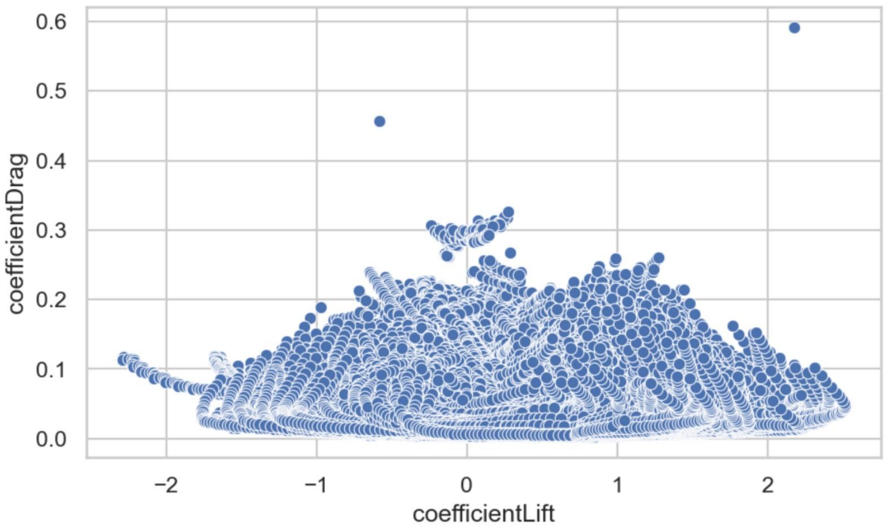


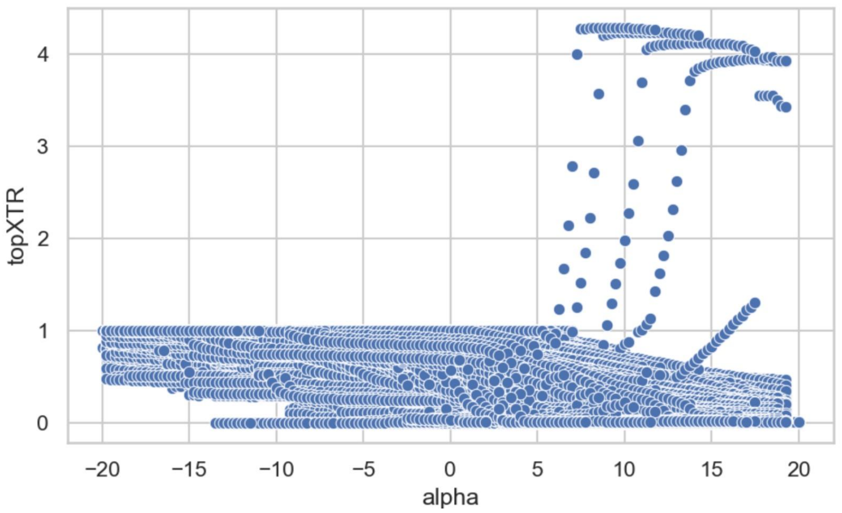
- The dataset contains values of different parameters such as angle of attack for a given reynolds number.
- The angle of attack vs reynolds number plot perfectly signifies the objective of our project, analysing the drag at various alpha values for a particular reynolds number so that we can identify the required orientation of the airfoil to achieve minimum drag at a given velocity.



- The Cl vs alpha distribution is approximately linear around the origin but there is a sharp drop in the Cl values at high angles of attack due to flow separation of air around the airfoil at high alpha values, which indicates the stall point of the airfoil.
- Also, when plotted individually, the symmetrical airfoils had their x-intercept quite close to the origin as expected while the cambered airfoils had their x-intercept skewed to the left of the origin as cambered airfoils have positive lift at zero angle of attack.

Priyanshi , 22b2213



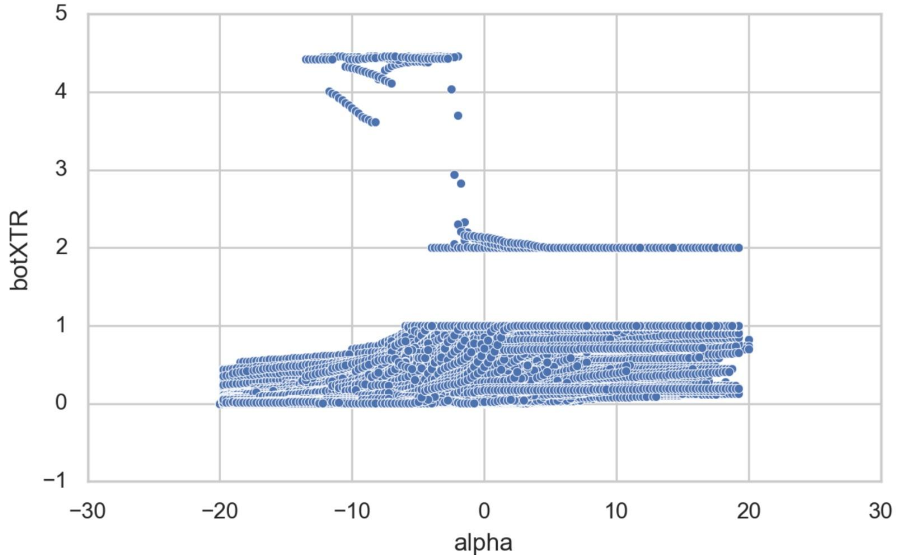


- The Cd vs Cl distribution, also known as the drag polar, is a parabolic curve which gives us some important airplane performance parameters such as Cd,min which is the minimum distance of the curve from the Cl axis.

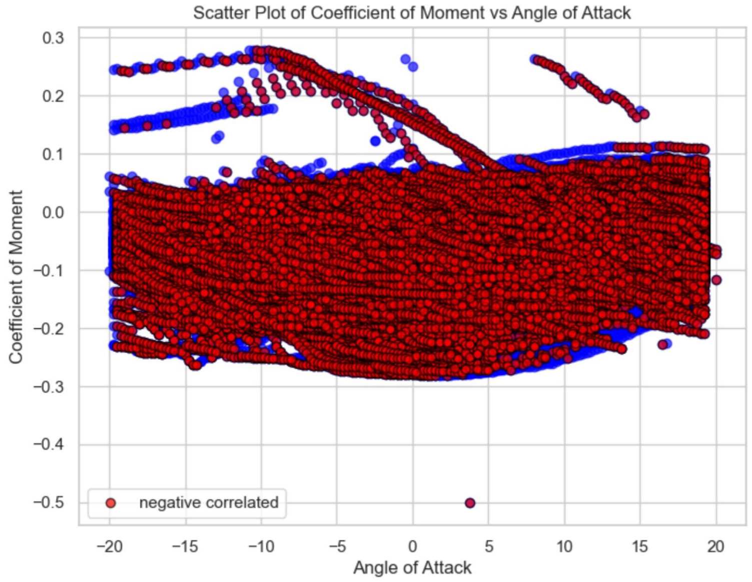
$$C_D = C_{D,\min} + \frac{(C_L - C_{L_{\min\ \text{drag}}})^2}{\pi e \mathrm{AR}}$$

- TopXTR is the normalized distance(distance divided by the chord of the airfoil) from the leading edge where transition occurs from laminar to turbulent flow on the upper surface of the airfoil.
- For most of the plots, as the value of angle of attack increases, flow separation starts to occur more closer to the leading edge. This is described by the decrease of topXTR as alpha increases.

Priyanshi , 22b2213

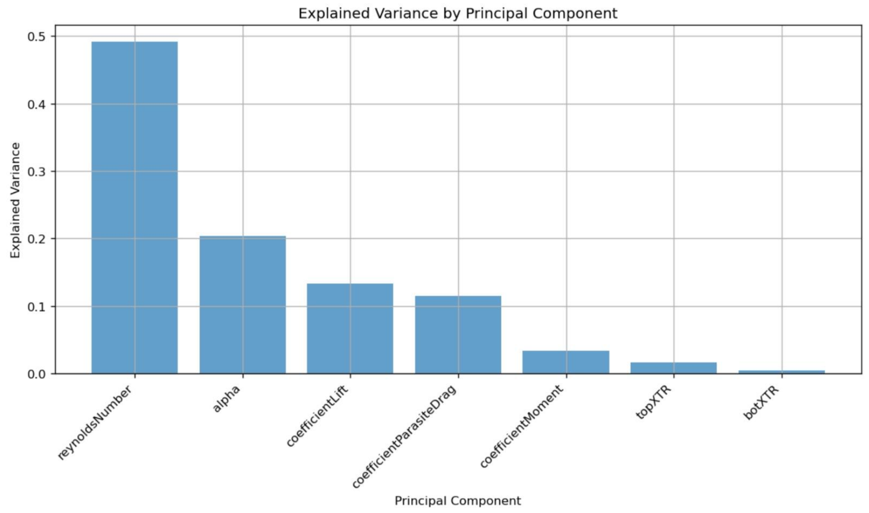


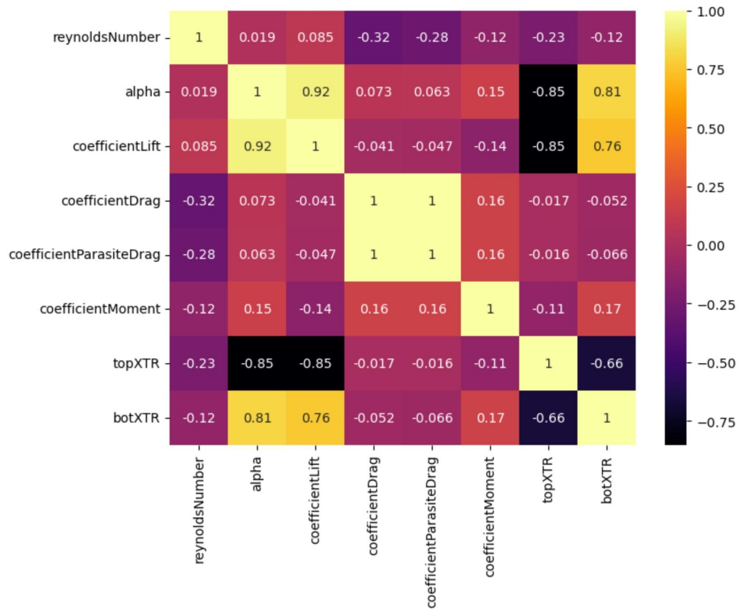
- BotXTR is the normalized distance(distance divided by the chord of the airfoil) from the leading edge where transition occurs from laminar to turbulent flow on the lower surface of the airfoil.
- Here, the opposite is observed with respect to the topXTR case. For most of the plots, as the value of angle of attack increases, flow separation starts to occur more further away from the leading edge. This is described by the increase of botXTR as alpha increases.



- An important stability prerequisite for an airfoil is its Cm vs alpha curve. To achieve positive stability, the Cm vs alpha plot should have a negative slope to allow the plane to self correct any perturbations in its pitch.
- We have highlighted all those plots which have a negative correlation, implying that only these airfoils will be positively stable.
- From this plot, we can conclude that our database almost entirely consists of positively stable airfoils.

Priyanshi , 22b2213





	reynoldsNumber	alpha	coefficientLift	coefficientDrag	coefficientParasiteDrag	coefficientMoment	topXTR	botXTR
reynoldsNumber	1	0.019	0.085	-0.32	-0.28	-0.12	-0.23	-0.12
alpha	0.019	1	0.92	0.073	0.063	0.15	-0.85	0.81
coefficientLift	0.085	0.92	1	-0.041	-0.047	-0.14	-0.85	0.76
coefficientDrag	-0.32	0.073	-0.041	1	1	0.16	-0.017	-0.052
coefficientParasiteDrag	-0.28	0.063	-0.047	1	1	0.16	-0.016	-0.066
coefficientMoment	-0.12	0.15	-0.14	0.16	0.16	1	-0.11	0.17
topXTR	-0.23	-0.85	-0.85	-0.017	-0.016	-0.11	1	-0.66
botXTR	-0.12	0.81	0.76	-0.052	-0.066	0.17	-0.66	1

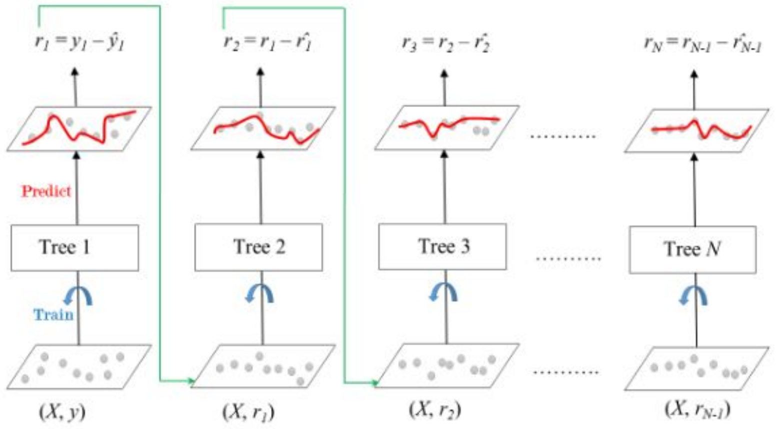
- This variance plot illustrates the relative significance or impact of various sets of parameters. As shown, Reynolds number has the highest significance in calculation the drag coefficient.
- The normalised distances (botXTR and topXTR) has close to zero variance meaning that their relevance is very low to our model.

- The Pearson's plot (heat map) aptly signifies the correlation between the various aerodynamic parameters.
- Cl and alpha have a high positive correlation of 0.92 as justified by their distribution shown earlier.
- Alpha also has a high inverse correlation with topXTR and a high direct correlation with botXTR which further verifies the plots obtained and the flow separation dynamics occurring on the airfoil.

Gradient Boosting Regression

Rohan Mekala , 22b2106

Gradient Boosting is a powerful boosting algorithm that combines several weak learners into strong learners, in which each new model is trained to minimize the loss function such as mean squared error or cross-entropy of the previous model using gradient descent.



Mathematics behind Gradient Boosting :-

- First, the loss function is computed.

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

- We then proceed to minimize the loss function L(f) with respect to f and if our gradient boosting algorithm is in M stages then we can improve the f_m algorithm by adding some new estimator as h_m where m lies between 1 and M.

$$\hat{y}_i = F_{m+1}(x_i) = F_m(x_i) + h_m(x_i)$$

- Assuming we are performing M stage gradient descent, the steepest gradient is calculated.

$$g_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$

- The current solution is computed by applying the gradient for the M trees.

$$f_m(x) = f_{m-1}(x) + \left(\underset{h_m \epsilon H}{argmin}\left[\sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + h_m(x_i))\right]\right)(x)$$

$$f_m = f_{m-1} - \rho_m g_m$$

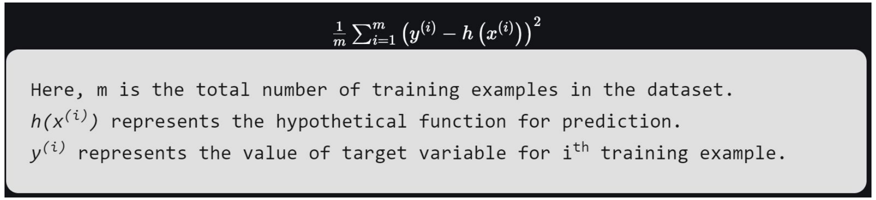
Justification for choice of model :-

- Gradient Boosting Regression can capture complex nonlinear relationships between features and the target variable (drag coefficient).
- It inherently mitigates the risk of overfitting, which is essential when working with limited datasets or noisy aerodynamic data.
- Gradient Boosting Regression implementation offers various hyperparameters such as the depth of individual trees, the learning rate, and the minimum sample leafs to fine-tune the model's performance.

Elastic Net Regression

Rohan Mekala , 22b2106

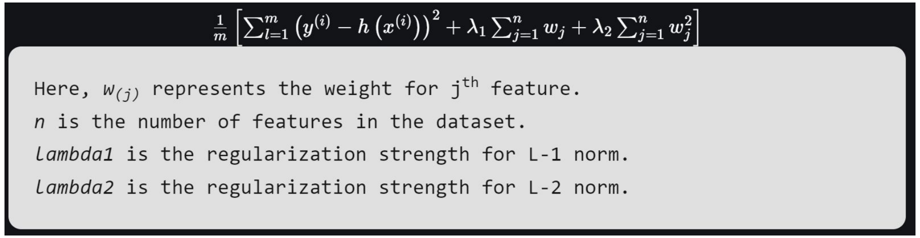
Elastic-Net Regression is a modification of Linear Regression which shares the same hypothetical function for prediction. The cost function of Linear Regression is represented by J.



$$\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} - h\left(x^{(i)}\right)\right)^2$$

Here, m is the total number of training examples in the dataset.
$h(x^{(i)})$ represents the hypothetical function for prediction.
$y^{(i)}$ represents the value of target variable for i$^{\text{th}}$ training example.

Linear Regression faces overfitting and struggles with collinear data, especially when dealing with numerous features, some irrelevant. To address this, Elastic-Net Regression combines L-2 and L-1 norm regularization, blending Ridge and Lasso benefits. It outperforms Lasso by offering better predictive power, feature selection, and hypothesis simplification. The modified cost function for Elastic-Net Regression is as follows:



$$\frac{1}{m}\left[\sum_{l=1}^{m}\left(y^{(i)} - h\left(x^{(i)}\right)\right)^2 + \lambda_1\sum_{j=1}^{n} w_j + \lambda_2\sum_{j=1}^{n} w_j^2\right]$$

Here, $w_{(j)}$ represents the weight for j$^{\text{th}}$ feature.
n is the number of features in the dataset.
lambda1 is the regularization strength for L-1 norm.
lambda2 is the regularization strength for L-2 norm.

Mathematics behind Elastic Net Regression :-

- The addition of an L-2 penalty term during gradient descent optimization reduces model weights toward zero, simplifying the hypothesis and mitigating overfitting.
- Incorporating an L-1 penalty further shrinks weights toward zero, effectively eliminating irrelevant features from the model.
- This weight penalization fosters a more predictive hypothesis, promoting sparsity (a model with fewer parameters).

Justification for choice of model :-

- In aerodynamic analysis, it's common to deal with correlated or multicollinear features, where some features are highly correlated with each other. Elastic Net Regression addresses this issue by simultaneously penalizing the absolute (L1) and squared (L2) magnitudes of the regression coefficients.
- Elastic Net Regression parameter alpha balances L1 and L2 regularization. At alpha = 1, it behaves like Lasso, promoting sparsity. At alpha = 0, it resembles Ridge, encouraging shrinkage without eliminating coefficients. By adjusting alpha, we can customize bias-variance trade-offs to suit our aerodynamic dataset.
- Elastic Net Regression, combining L1 and L2 penalties, is robust to outliers. L2 penalty (Ridge) reduces outlier influence by shrinking coefficients, while L1 penalty (Lasso) promotes sparsity and feature selection.

Linear Regression

Rohan Mekala , 22b2106

- Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation to observed data.
- We will be focusing on using multiple linear regression which includes more than one independent variable and one dependent variable. The equation for multiple linear regression is:

$$y = \beta_0 + \beta_1 X + \beta_2 X + \ldots\ldots\ldots \beta_n X$$

- In Linear Regression, the Mean Squared Error (MSE) function calculates squared errors between predicted values $\hat{y}_i$ and actual values y_i. It aims to find optimal values for the intercept (theta1) and coefficient of the input feature (theta2) to fit a line to the data.

$$\text{Cost function}(J) = \frac{1}{n}\sum_n^i (\hat{y}_i - y_i)^2$$

Mathematics behind Linear Regression :-

- Suppose there's a variable Y_i. The distance between Y_i and the predicted value is what is called sum of squared estimate of errors or SSE. This is the unexplained variance and it must be minimized to get the best accuracy.

$$\mathrm{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Calculation of coefficients of the linear relation :-

$$\beta_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} \left(x_i - \bar{x}\right)^2}$$

Justification for choice of model :-

- Linear Regression serves as a natural baseline model for evaluating the performance of more complex algorithms.
- Linear Regression offers clear insights into the relationships between input features and the target variable (drag coefficient). Coefficients indicate both the magnitude and direction of feature effects, aiding aerodynamic analysis by revealing how changes in airfoil parameters impact drag coefficient.
- Training a linear regression model typically involves solving a system of linear equations, which can be done efficiently using matrix operations. This efficiency is advantageous when dealing with large volumes of aerodynamic data,

XGBoost Regression

Rohan Mekala , 22b2106

XGBoost is an algorithm used for both regression and classification tasks. XGBoost is an ensemble learning method that builds a series of decision trees sequentially, where each tree corrects the errors made by the previous ones, also known as gradient boosting. XGBoost builds a predictive model by combining the predictions of multiple individual models, often decision trees, in an iterative manner.

$$obj(\theta) = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

- XGBoost includes both L1 (Lasso) and L2 (Ridge) regularization terms to control the complexity of the model and prevent overfitting.

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$\text{Loss}(\hat{y}_i, y_i) = (y_i - \hat{y}_i)^2.$$

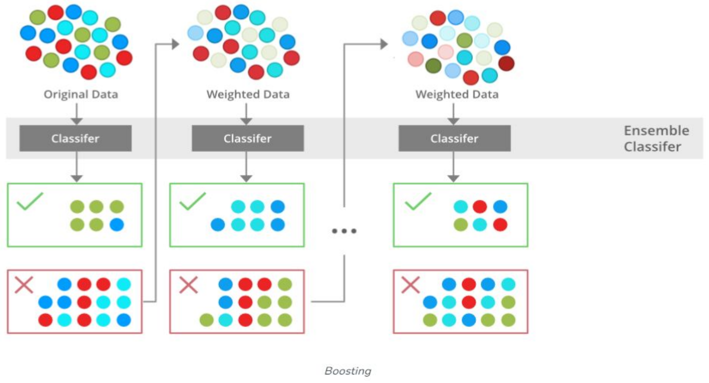
- In each iteration, XGBoost calculates the negative gradient of the loss function with respect to the predicted values, denoted as

$$g_i = \frac{\partial \text{Loss}(\hat{y}_i, y_i)}{\partial \hat{y}_i}$$

- It also computes the second derivative of the loss function, denoted as

$$h_i = \frac{\partial^2 \text{Loss}(\hat{y}_i, y_i)}{\partial \hat{y}_i^2}$$

- These gradients are used in the optimization process to update the model parameters.



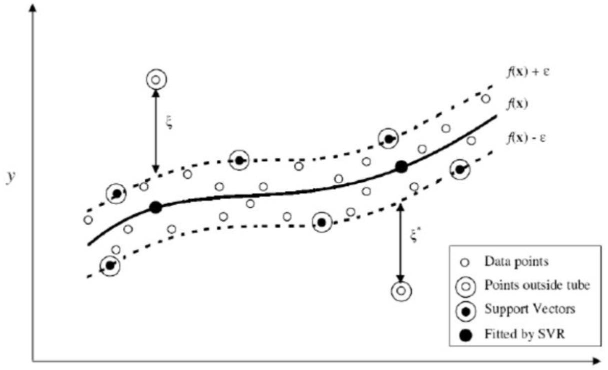
Justification for choice of model :-

- XGBoost is highly scalable and efficient, making it suitable for handling large datasets commonly encountered in aerodynamic analysis.
- It assigns missing values to the direction that minimizes the loss function, effectively incorporating information from incomplete data points.
- XGBoost employs tree pruning techniques such as depth-based and leaf-wise growth to control the complexity of individual trees. Depth-based tree growth limits the maximum depth of each tree, preventing overfitting and reducing the risk of memorizing noise in the data.

Support Vector Regression

Varad Patil, 22b2270

Support Vector Regression (SVR) is a technique used in machine learning to predict continuous values. It does this by drawing a line (or plane if there are many features) that best matches the training data while allowing some room for points to be a bit away from the line. SVR looks for special points called support vectors that guide where the line should go. It tries to make the line fit most of the points well while also not worrying too much about every single point. It's a useful method for making predictions, especially when dealing with complex data patterns and noise.



$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

- SVR optimizes the model parameters (weights *w* and bias *b*) by minimizing the loss function subject to the margin constraints.

$$\text{Loss} = \frac{1}{2}\left(||w||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)\right)$$

$$y_i - (\hat{y}_i + b) \leq \epsilon + \xi_i$$

$$(\hat{y}_i + b) - y_i \leq \epsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

- SVR has hyperparameters that need to be tuned for optimal performance, such as the regularization parameter (C), epsilon (margin width), and the choice of kernel function..

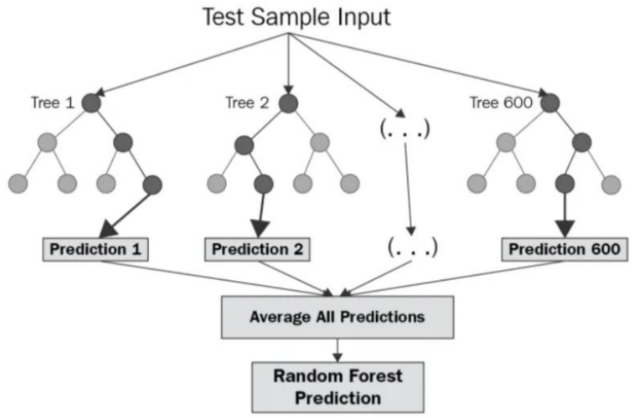
Justification for choice of model :-

- Support Vector Regression captures nonlinear relationships between input features and the target variable (drag coefficient) by mapping them into a higher-dimensional space using a kernel function like radial basis function kernel, potentially making the data linearly separable.
- SVR's epsilon-insensitive loss function makes it robust to outliers in training data by penalizing only errors beyond a certain threshold (epsilon), reducing sensitivity compared to traditional least squares regression methods.

Random Forest Regression

Varad Patil, 22b2270

- A Random Forest Regressor is a ML algorithm that excels in predicting continuous numerical values. It operates by constructing an ensemble of decision trees, each trained on a random subset of the training data and a random subset of features.
- This diversity among trees helps prevent overfitting and enhances the model's ability to generalize well to new, unseen data. During prediction, each decision tree provides an estimate, and the final prediction from the Random Forest Regressor is typically the average (or median) of these individual tree predictions.
- This aggregation of predictions reduces variance and improves the overall accuracy of the model.



- By tuning hyperparameters such as the number of trees and maximum tree depth, Random Forest Regressors can be customized to achieve optimal performance for a wide range of regression tasks, making them a valuable tool in machine learning.

$$\hat{y} = \frac{1}{N}\sum_{i=1}^{N}\hat{y}_i$$

Justification for choice of model :-

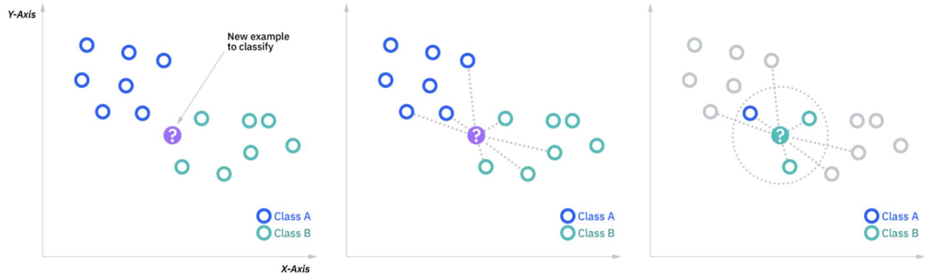
- Random Forest Regression is robust to overfitting. By aggregating predictions from multiple trees and utilizing techniques like bagging and feature randomization, it mitigates overfitting risks. This ensures reliable predictions of drag coefficients across diverse airfoil configurations and conditions.
- Unlike traditional linear regression models, which may struggle with correlated predictors, Random Forest Regression can handle multicollinear features effectively. This allows it to identify and exploit interactions between airfoil parameters, leading to improved predictive performance.
- Since each decision tree in the ensemble is trained independently on a random subset of data samples, outliers are less likely to have a significant impact on the overall model predictions.

Nearest Neighbors Regression

Varad Patil, 22b2270

The **k-nearest neighbors** (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point into k groups .

The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.



K affects the accuracy of our model and the number of nearest neighbors from which out new data is classified is highly significant very small value for “k” could lead to overfitting and very small value for k could lead to underfitting. So, Choosing the right value for K is very important

Loss Function for Regression : We minimize the MSE to get better accuracy

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Mean Squared Error

Hyperparameters : k - The number of nearest neighbors , Distance metrics for eg. the popular one are Euclidean, Minkowski and Manhattan distance .

$$\text{Manhattan Distance} = d(x,y) = \left(\sum_{i=1}^{m}|x_i - y_i|\right)$$

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

$$\text{Minkowski Distance} = \left(\sum_{i=1}^{n}|x_i - y_i|\right)^{1/p}$$

This algorithm calculates the distance between data-points , so it is highly affected by the scale of feature. If one feature scale is very high as compared to others ,other distance gets automatically neglected. Thus Standardization / Normalization methods are used.

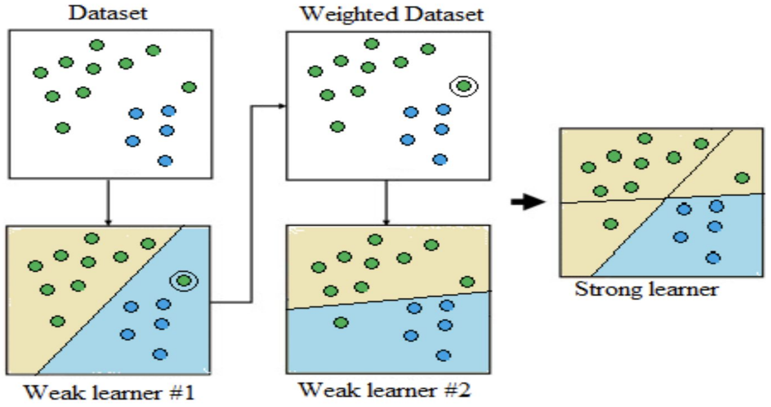
Justification for choice of model :-

- As from the initial data analysis , the data is non-linearly distributed.
- k nearest neighbour algorithm is useful when the data is non-linear and relationship between predictors and independent variable is complex .
- It does not assume any underlying distribution and captures the local patterns of data .

AdaBoost Regression

Varad Patil, 22b2270

An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.



Loss function for regression :

The loss function that AdaBoost uses is an **exponential loss function** where Cm(xi) is the predicted output.

$$E_m = \sum_{i=1}^{N} e^{-y_i C_m(x_i)}$$

Derivative of Loss function :

$$E_m = \sum_{y_i \neq K_m(x_i)} w_i^{(m)} e^{\alpha_m} + \sum_{y_i = K_m(x_i)} w_i^{(m)} e^{-\alpha_m}$$

Hyperparameters : no of estimators (No of weak learners (Decision Tree)), learning rate , loss function like “linear” , “square” and “exponential”

Justification for choice of model :-

- Adaboost is effective in improving predictive accuracy and reducing bias .
- It is particularly useful when there are complex relationships between predictors and response variable as it focuses on instances that are difficult to predict .
- Adaboost regressor is less prone to overfitting compared to Individual weak learners, especially when using shallow decision trees as base estimators .

Conclusions

- By performing data analysis using various plots , we found out that most of the features are showing trends which are consistent with real life aerodynamics.
- For achieving stability, the data points between “Cm” and “α” need to have a negative slope which in our case was shown by majority of the points.
- For different α’s , topXTR and botXTR should have values ≤ 1. In our case ,most of the points (around ~99.7% of the total data) were in this range which is an indication of a good and realistic data .
- Using different plots between features, we found out that there is a non-linear relationship between features, which is a useful piece of information and can be used for model selections that perform better on this type of data-set.
- We are using 8 different models, and the choice of these models was made due to the fact that, the data is non-linear and by observing the size of the data, we needed models which could handle bigger datasets. We are also using 1-2 simple models which can act as baseline models for a comparative analysis.
- Most of these models are prone to overfitting and have less bias during predictions. By performing Principal Component Analysis (PCA), we were able to show which feature explains the maximum variations of the data. In our case, reynolds number was the one to show the maximum variance of around ~49%, which is coherent with the fact that the reynolds number is a very important non-dimensional term in aerodynamics.

final-stage3

May 4, 2024

1 Final Project Program :

2 Priyanshi , 22B2213

3 Rohan Mekala , 22B2106

4 Varad Patil , 22B2270

5 ME228 - Project , Group-10

6 *Aerodynamic Drag Prediction and Comparative Analysis of Models*

7 Importing essential libraries

[17]:
```
pip install statsmodels
```

```
Collecting statsmodelsNote: you may need to restart the kernel to use updated
packages.

  Downloading statsmodels-0.14.2-cp311-cp311-win_amd64.whl.metadata (9.5 kB)
Requirement already satisfied: numpy>=1.22.3 in
c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from
statsmodels) (1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in
c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from
statsmodels) (1.12.0)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in
c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from
statsmodels) (2.2.0)
Collecting patsy>=0.5.6 (from statsmodels)
  Downloading patsy-0.5.6-py2.py3-none-any.whl.metadata (3.5 kB)
Requirement already satisfied: packaging>=21.3 in
c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from
statsmodels) (23.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from
pandas!=2.1.0,>=1.4->statsmodels) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from
pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from
pandas!=2.1.0,>=1.4->statsmodels) (2023.4)
Requirement already satisfied: six in
c:\users\hp\appdata\local\programs\python\python311\lib\site-packages (from
patsy>=0.5.6->statsmodels) (1.16.0)
Downloading statsmodels-0.14.2-cp311-cp311-win_amd64.whl (9.9 MB)
   ---------------------------------------- 0.0/9.9 MB ? eta -:--:--
    --------------------------------------- 0.2/9.9 MB 5.8 MB/s eta 0:00:02
   --- ------------------------------------ 0.8/9.9 MB 10.0 MB/s eta 0:00:01
   ------ --------------------------------- 1.5/9.9 MB 11.9 MB/s eta 0:00:01
   ---------- ----------------------------- 2.7/9.9 MB 15.4 MB/s eta 0:00:01
   ---------------- ----------------------- 4.0/9.9 MB 18.3 MB/s eta 0:00:01
   --------------------- ------------------ 5.4/9.9 MB 20.2 MB/s eta 0:00:01
   --------------------------- ------------ 6.8/9.9 MB 21.6 MB/s eta 0:00:01
   --------------------------------- ------ 8.2/9.9 MB 22.8 MB/s eta 0:00:01
   ---------------------------------------  9.6/9.9 MB 23.7 MB/s eta 0:00:01
   ---------------------------------------- 9.9/9.9 MB 22.5 MB/s eta 0:00:00
Downloading patsy-0.5.6-py2.py3-none-any.whl (233 kB)
   ---------------------------------------- 0.0/233.9 kB ? eta -:--:--
   ---------------------------------------- 233.9/233.9 kB ? eta 0:00:00
Installing collected packages: patsy, statsmodels
Successfully installed patsy-0.5.6 statsmodels-0.14.2
```

[4]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
```

8 Reading the dataset

[5]:
```python
df = pd.read_csv("combinedAirfoilDataLabeled.csv")
```

```
/var/folders/pq/sld6wjhs6lgg86m1qhqkwrzh0000gn/T/ipykernel_8961/4210192359.py:1:
DtypeWarning: Columns (70) have mixed types. Specify dtype option on import or
set low_memory=False.
  df = pd.read_csv("combinedAirfoilDataLabeled.csv")
```

9 Cleaning the data

[6]:
```
index_to_drop = df[df['botXTR'] == ' 0.0095.1'].index
df.drop(index_to_drop, inplace=True)
df['botXTR'] = df['botXTR'].astype(float)
```

10 Creating Dependent and Independent Variables

[7]:
```
# Assuming 'df' is your DataFrame
column_names = df.
 ↪drop(columns=['coefficientDrag','airfoilName','upperSurfaceCoeff1',␣
 ↪'upperSurfaceCoeff2',
       'upperSurfaceCoeff3', 'upperSurfaceCoeff4', 'upperSurfaceCoeff5',
       'upperSurfaceCoeff6', 'upperSurfaceCoeff7', 'upperSurfaceCoeff8',
       'upperSurfaceCoeff9', 'upperSurfaceCoeff10', 'upperSurfaceCoeff11',
       'upperSurfaceCoeff12', 'upperSurfaceCoeff13', 'upperSurfaceCoeff14',
       'upperSurfaceCoeff15', 'upperSurfaceCoeff16', 'upperSurfaceCoeff17',
       'upperSurfaceCoeff18', 'upperSurfaceCoeff19', 'upperSurfaceCoeff20',
       'upperSurfaceCoeff21', 'upperSurfaceCoeff22', 'upperSurfaceCoeff23',
       'upperSurfaceCoeff24', 'upperSurfaceCoeff25', 'upperSurfaceCoeff26',
       'upperSurfaceCoeff27', 'upperSurfaceCoeff28', 'upperSurfaceCoeff29',
       'upperSurfaceCoeff30', 'upperSurfaceCoeff31', 'lowerSurfaceCoeff1',
       'lowerSurfaceCoeff2', 'lowerSurfaceCoeff3', 'lowerSurfaceCoeff4',
       'lowerSurfaceCoeff5', 'lowerSurfaceCoeff6', 'lowerSurfaceCoeff7',
       'lowerSurfaceCoeff8', 'lowerSurfaceCoeff9', 'lowerSurfaceCoeff10',
       'lowerSurfaceCoeff11', 'lowerSurfaceCoeff12', 'lowerSurfaceCoeff13',
       'lowerSurfaceCoeff14', 'lowerSurfaceCoeff15', 'lowerSurfaceCoeff16',
       'lowerSurfaceCoeff17', 'lowerSurfaceCoeff18', 'lowerSurfaceCoeff19',
       'lowerSurfaceCoeff20', 'lowerSurfaceCoeff21', 'lowerSurfaceCoeff22',
       'lowerSurfaceCoeff23', 'lowerSurfaceCoeff24', 'lowerSurfaceCoeff25',
       'lowerSurfaceCoeff26', 'lowerSurfaceCoeff27', 'lowerSurfaceCoeff28',
       'lowerSurfaceCoeff29', 'lowerSurfaceCoeff30',␣
 ↪'lowerSurfaceCoeff31'],axis=1).columns

# Separate the features (X) from the target variable if applicable
X = df.drop(columns=['coefficientDrag','airfoilName','upperSurfaceCoeff1',␣
 ↪'upperSurfaceCoeff2',
       'upperSurfaceCoeff3', 'upperSurfaceCoeff4', 'upperSurfaceCoeff5',
       'upperSurfaceCoeff6', 'upperSurfaceCoeff7', 'upperSurfaceCoeff8',
       'upperSurfaceCoeff9', 'upperSurfaceCoeff10', 'upperSurfaceCoeff11',
       'upperSurfaceCoeff12', 'upperSurfaceCoeff13', 'upperSurfaceCoeff14',
       'upperSurfaceCoeff15', 'upperSurfaceCoeff16', 'upperSurfaceCoeff17',
       'upperSurfaceCoeff18', 'upperSurfaceCoeff19', 'upperSurfaceCoeff20',
       'upperSurfaceCoeff21', 'upperSurfaceCoeff22', 'upperSurfaceCoeff23',
       'upperSurfaceCoeff24', 'upperSurfaceCoeff25', 'upperSurfaceCoeff26',
       'upperSurfaceCoeff27', 'upperSurfaceCoeff28', 'upperSurfaceCoeff29',
       'upperSurfaceCoeff30', 'upperSurfaceCoeff31', 'lowerSurfaceCoeff1',
       'lowerSurfaceCoeff2', 'lowerSurfaceCoeff3', 'lowerSurfaceCoeff4',
       'lowerSurfaceCoeff5', 'lowerSurfaceCoeff6', 'lowerSurfaceCoeff7',
       'lowerSurfaceCoeff8', 'lowerSurfaceCoeff9', 'lowerSurfaceCoeff10',
       'lowerSurfaceCoeff11', 'lowerSurfaceCoeff12', 'lowerSurfaceCoeff13',
       'lowerSurfaceCoeff14', 'lowerSurfaceCoeff15', 'lowerSurfaceCoeff16',
       'lowerSurfaceCoeff17', 'lowerSurfaceCoeff18', 'lowerSurfaceCoeff19',
       'lowerSurfaceCoeff20', 'lowerSurfaceCoeff21', 'lowerSurfaceCoeff22',
       'lowerSurfaceCoeff23', 'lowerSurfaceCoeff24', 'lowerSurfaceCoeff25',
       'lowerSurfaceCoeff26', 'lowerSurfaceCoeff27', 'lowerSurfaceCoeff28',
       'lowerSurfaceCoeff29', 'lowerSurfaceCoeff30', 'lowerSurfaceCoeff31'])  #␣
↪Remove the target column if applicable
```

[8]:
```
X
```

[8]:
```
        reynoldsNumber  alpha  coefficientLift  coefficientParasiteDrag  \
0                50000  -8.25          -0.2446                  0.10788
1                50000  -8.00          -0.2475                  0.10812
2                50000  -7.75          -0.2530                  0.10902
3                50000  -7.50          -0.2326                  0.10052
4                50000  -7.25          -0.2270                  0.09765
…                  …      …                …                        …
867093         1000000   7.25           1.0045                  0.01933
867094         1000000   7.50           1.0247                  0.02070
867095         1000000   7.75           1.0441                  0.02198
867096         1000000  16.75           0.8792                  0.20941
867097         1000000  17.00           0.8857                  0.21347

        coefficientMoment  topXTR  botXTR
0                 -0.0201  1.0000  0.0956
1                 -0.0218  1.0000  0.0969
2                 -0.0240  1.0000  0.0975
3                 -0.0214  1.0000  0.1007
4                 -0.0208  1.0000  0.1042
…                     …       …       …
867093            -0.0693  0.0036  1.0000
867094            -0.0679  0.0037  1.0000
867095            -0.0663  0.0038  1.0000
867096            -0.0874  0.0036  1.0000
867097            -0.0894  0.0036  1.0000

[867097 rows x 7 columns]
```

[9]:
```
y = df['coefficientDrag']
```

[10]:
```
y
```

```
[10]: 0          0.11407
      1          0.11417
      2          0.11491
      3          0.10640
      4          0.10345
                  …
      867093     0.02441
      867094     0.02564
      867095     0.02678
      867096     0.21102
      867097     0.21509
      Name: coefficientDrag, Length: 867097, dtype: float64
```

---

11 Varad Patil , 22B22270

12 1. Linear Regression

```
[11]: import statsmodels.api as stm
      from sklearn.model_selection import train_test_split
```

12.1 Splitting the data into train and test split

```
[12]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,␣
       ↪random_state=7)
```

12.2 Scaling the data

```
[13]: scaler = StandardScaler()
```

```
[14]: scaled_xtrain = scaler.fit_transform(X_train)
      scaled_xtest = scaler.transform(X_test)
```

```
[15]: scaled_xTrain = stm.add_constant(scaled_xtrain)
      scaled_xTest = stm.add_constant(scaled_xtest)
```

12.3 Training the model on train data

```
[16]: # Training the linear regression model
      model = stm.OLS(y_train, scaled_xTrain)
      results = model.fit()
```

12.4 Displaying the linear regression summary with different parameters

[17]:
```python
# Printing the summary of the model
print(results.summary())
```

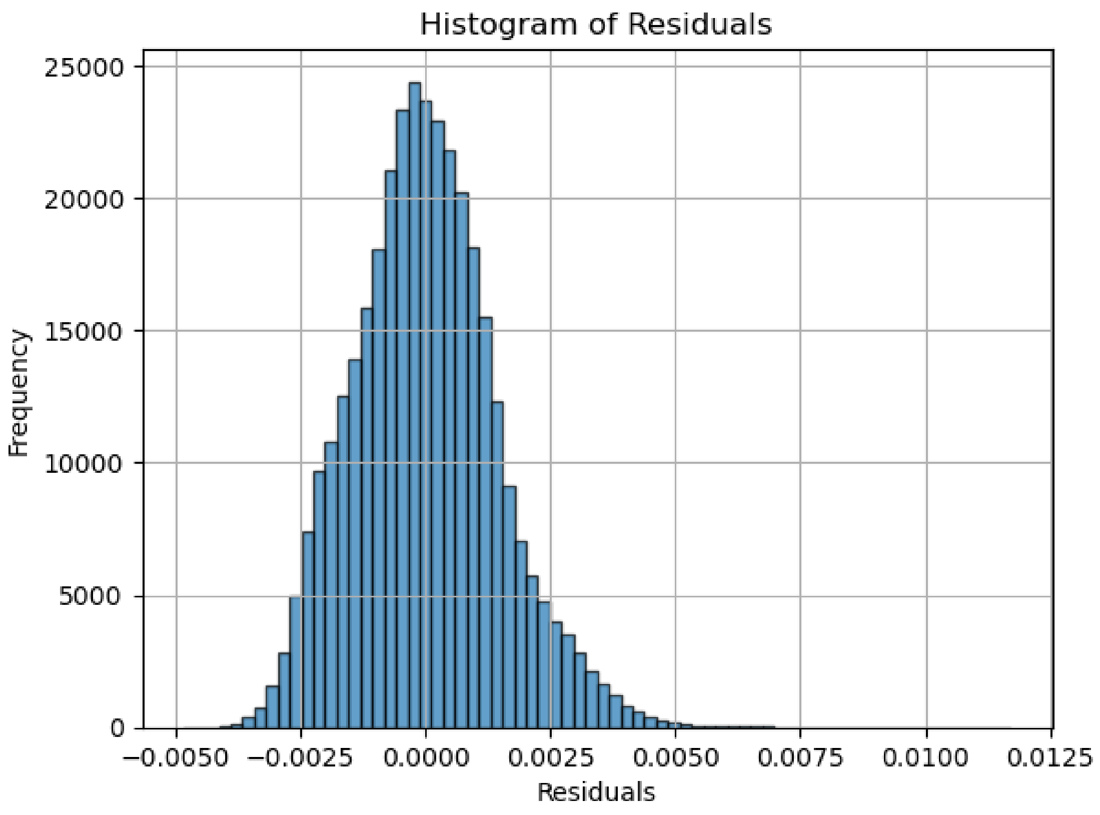
```
                            OLS Regression Results
==============================================================================
Dep. Variable:        coefficientDrag   R-squared:                       0.998
Model:                            OLS   Adj. R-squared:                  0.998
Method:                 Least Squares   F-statistic:                 3.843e+07
Date:                Sat, 04 May 2024   Prob (F-statistic):               0.00
Time:                        20:44:57   Log-Likelihood:             2.6675e+06
No. Observations:              520258   AIC:                        -5.335e+06
Df Residuals:                  520250   BIC:                        -5.335e+06
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0384   1.99e-06   1.93e+04      0.000       0.038       0.038
x1            -0.0014   2.37e-06   -572.917      0.000      -0.001      -0.001
x2             0.0011   7.74e-06    140.118      0.000       0.001       0.001
x3            -0.0011   8.52e-06   -131.484      0.000      -0.001      -0.001
x4             0.0322   2.23e-06   1.44e+04      0.000       0.032       0.032
x5            -0.0003   3.14e-06   -107.888      0.000      -0.000      -0.000
x6            -0.0004    4.6e-06    -88.721      0.000      -0.000      -0.000
x7          1.194e-05   3.69e-06      3.238      0.001    4.71e-06    1.92e-05
==============================================================================
Omnibus:                    17930.728   Durbin-Watson:                   2.002
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            22383.419
Skew:                           0.400   Prob(JB):                         0.00
Kurtosis:                       3.626   Cond. No.                         10.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

12.5 Predictions using test data

[18]:
```python
# Making predictions on the test set
predictions = results.predict(scaled_xTest)
```

[19]:
```python
lr_residuals = y_test - predictions
```

[20]:
```python
plt.hist(lr_residuals, bins=70, edgecolor='black', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



```
[21]: from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error
```

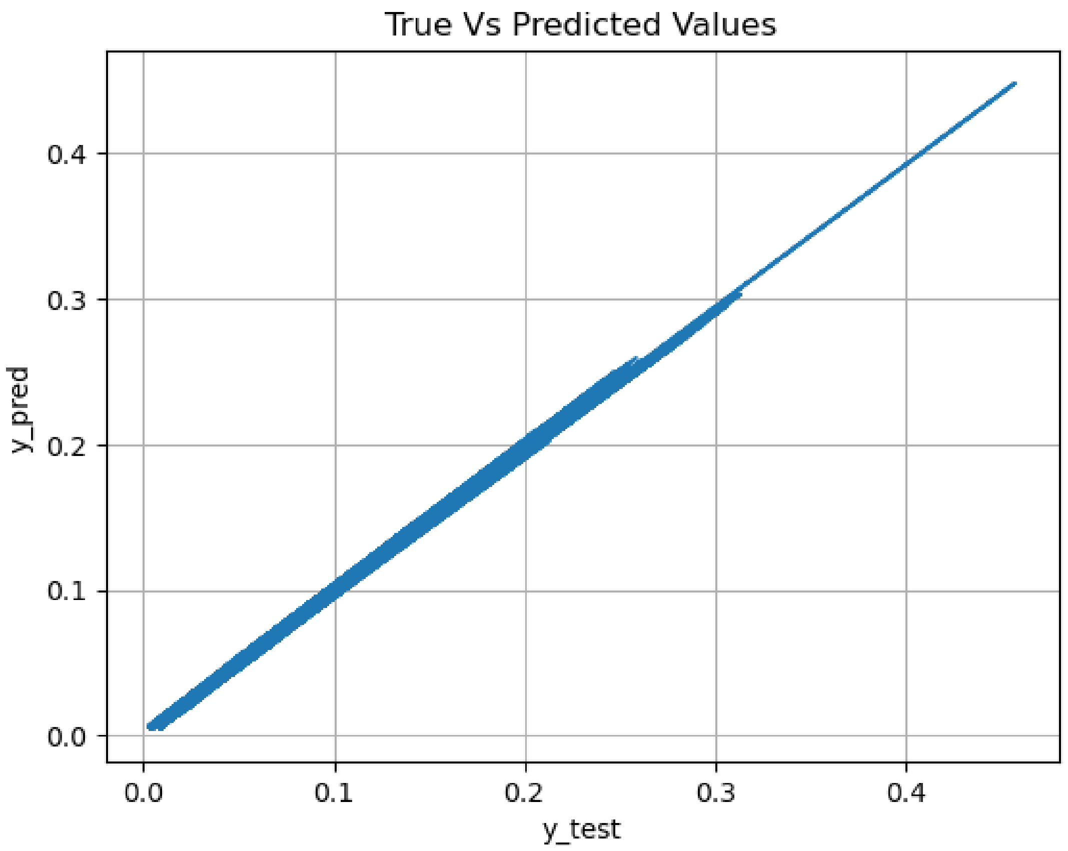
12.6 Metric Evaluations

```
[22]: print("Mean Squared Error:", mean_squared_error(y_test, predictions))
print("RMSE:",np.sqrt(mean_squared_error(y_test,predictions)))
print("R^2 Score:", r2_score(y_test, predictions))
```

```
Mean Squared Error: 2.063141244569127e-06
RMSE: 0.0014363638969875033
R^2 Score: 0.9980906823111286
```

12.7 Plot of True values vs Predicted values

[23]:
```
plt.plot(y_test,predictions)
plt.grid()
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.title("True Vs Predicted Values");
```



12.8 Remarks:

- Durbin-Watson checks whether there's a pattern in residuals (y-y_pred). 2 means **no auto-correlation** ie no pattern in residuals. In our case also , value is 2.002 ~ 2.
- Condition Number tells, how sensitive the model is to new data. **Cond.No. < 10** , is good . In our case also almost 10 thus less sensitive model.

- The residuals of a regression analysis are **normally distributed**, it suggests that the model accurately captures the relationship between the variables, meeting the assumptions for many statistical tests and confidence intervals.
- The plot between y_test and y_pred should show a strong linear relationship between y_test and y_pred. In our plot the points cluster closely around the 45-degree line, indicating a strong linear relationship between actual and predicted values.Thus, a good model.

13 2. Elastic Net

```
[24]: from sklearn.linear_model import ElasticNetCV
```

```
[25]: elastic_model = ElasticNetCV(l1_ratio=[.1, .5, .7,.9, .95, .99, 1],tol=0.01)
```

13.1 Training the model on train data

```
[26]: elastic_model.fit(scaled_xTrain,y_train)
```

```
[26]: ElasticNetCV(l1_ratio=[0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1], tol=0.01)
```

```
[27]: elastic_model.l1_ratio_
```

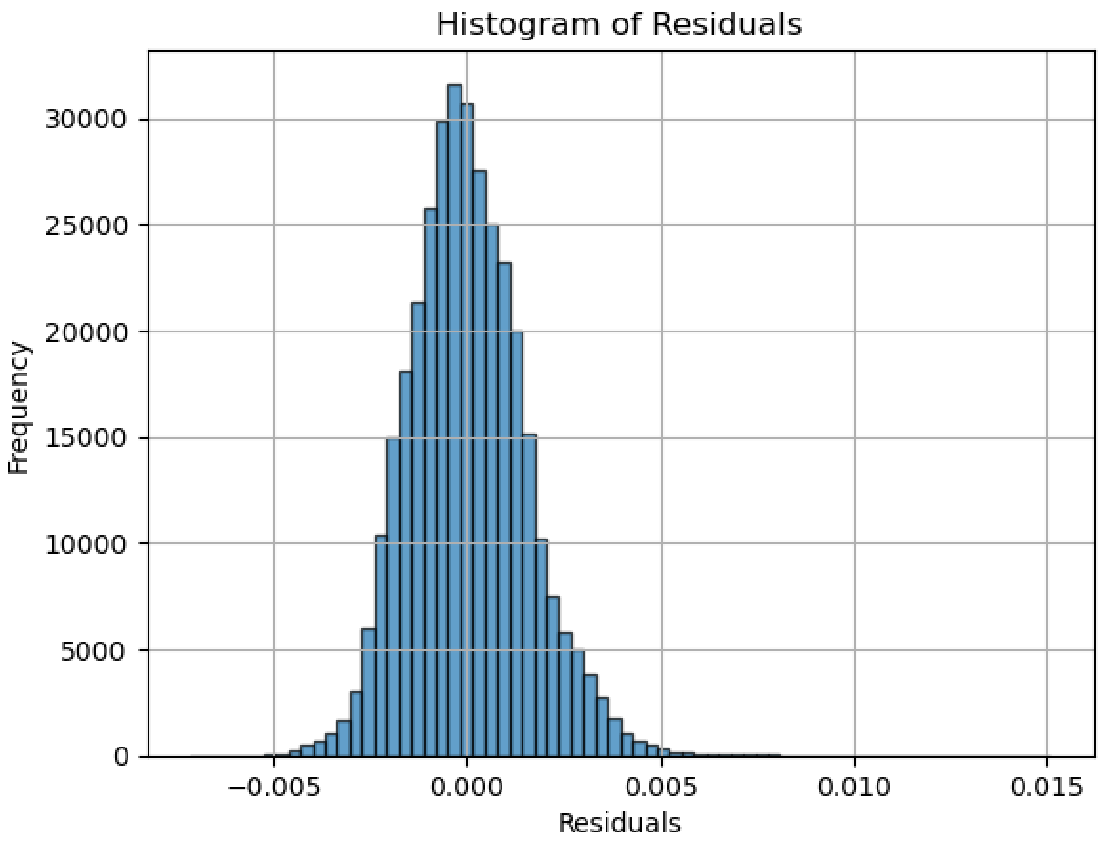
```
[27]: 1.0
```

13.2 Predictions using test data

```
[28]: test_predictions = elastic_model.predict(scaled_xTest)
```

```
[29]: el_residuals = y_test - test_predictions
```

```
[30]: plt.hist(el_residuals, bins=70, edgecolor='black', alpha=0.7)
      plt.title('Histogram of Residuals')
      plt.xlabel('Residuals')
      plt.ylabel('Frequency')
      plt.grid(True)
      plt.show()
```



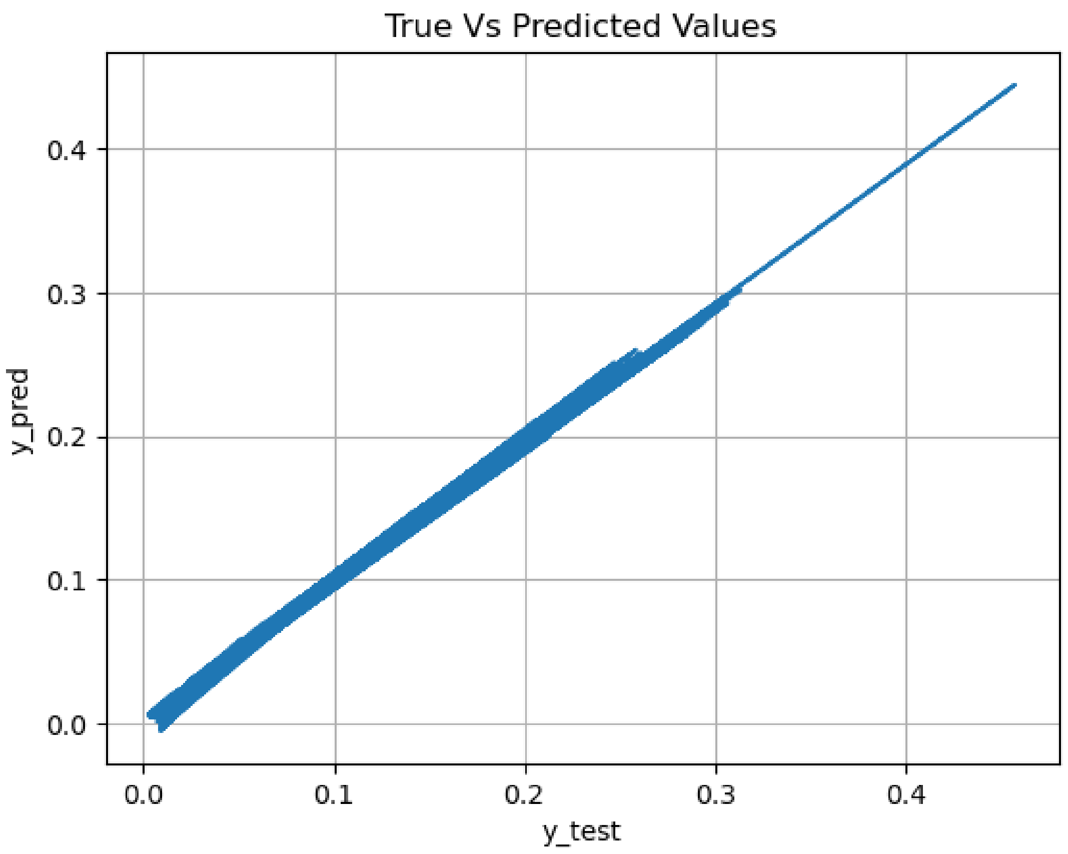
13.3 Metric Evaluations

[31]:
```
print("Mean Squared Error:", mean_squared_error(y_test, test_predictions))
print("RMSE:",np.sqrt(mean_squared_error(y_test,test_predictions)))
print("R^2 Score:", r2_score(y_test, test_predictions))
```

```
Mean Squared Error: 2.2966225477783885e-06
RMSE: 0.0015154611667008786
R^2 Score: 0.9978746088923011
```

13.4 Plot of True values vs Predicted values

[32]:
```
plt.plot(y_test,test_predictions)
plt.grid()
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.title("True Vs Predicted Values");
```



13.5 Remarks :

- The residuals of a regression analysis are normally distributed, it suggests that the model accurately captures the relationship between the variables, meeting the assumptions for many statistical tests and confidence intervals.
- Here it is a good model, as the points to cluster closely around the 45-degree line, indicating a strong linear relationship between actual and predicted values.

14 3. Random Forest Regressor

[33]:
```
from sklearn.ensemble import RandomForestRegressor
```

[34]:
```
rf_model = RandomForestRegressor()
```

14.1 Training model on train data

[35]:
```
rf_model.fit(X_train,y_train)
```

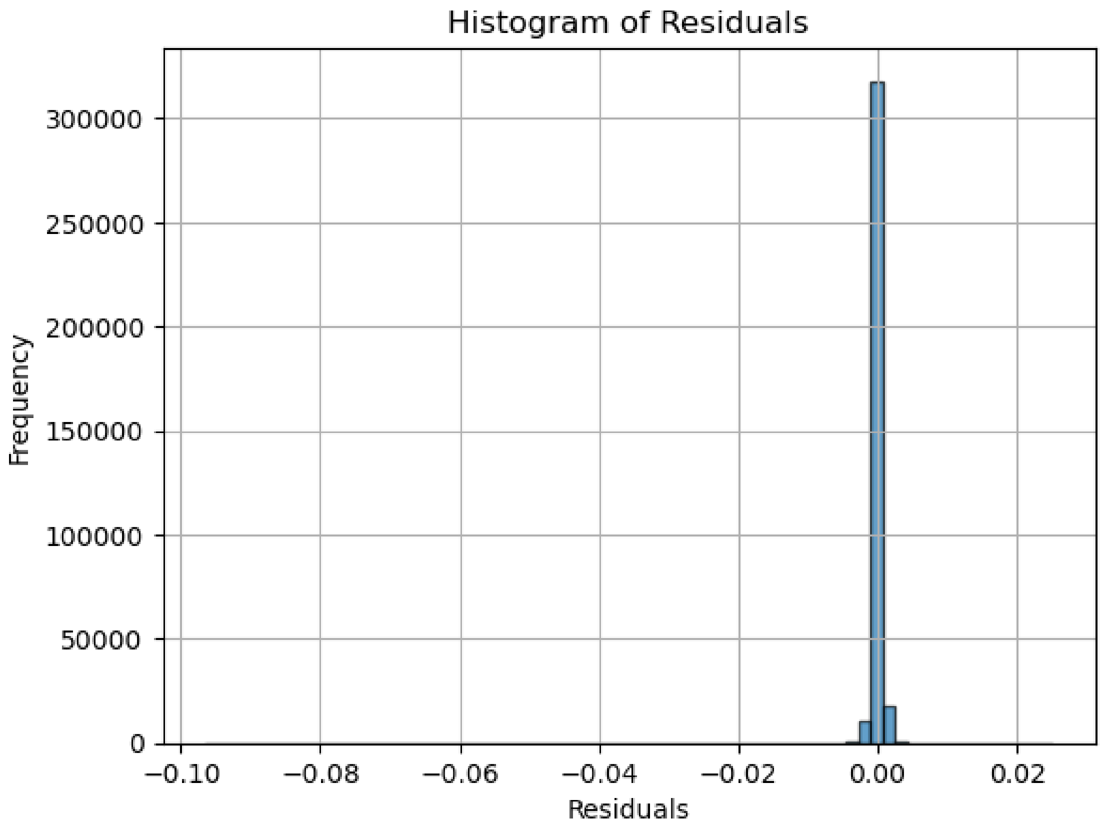
[35]:
```
RandomForestRegressor()
```

14.2 Predictions on test data

[36]:
```
rf_pred = rf_model.predict(X_test)
```

[37]:
```
rf_residuals = y_test - rf_pred
```

[38]:
```
plt.hist(rf_residuals, bins=70, edgecolor='black', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

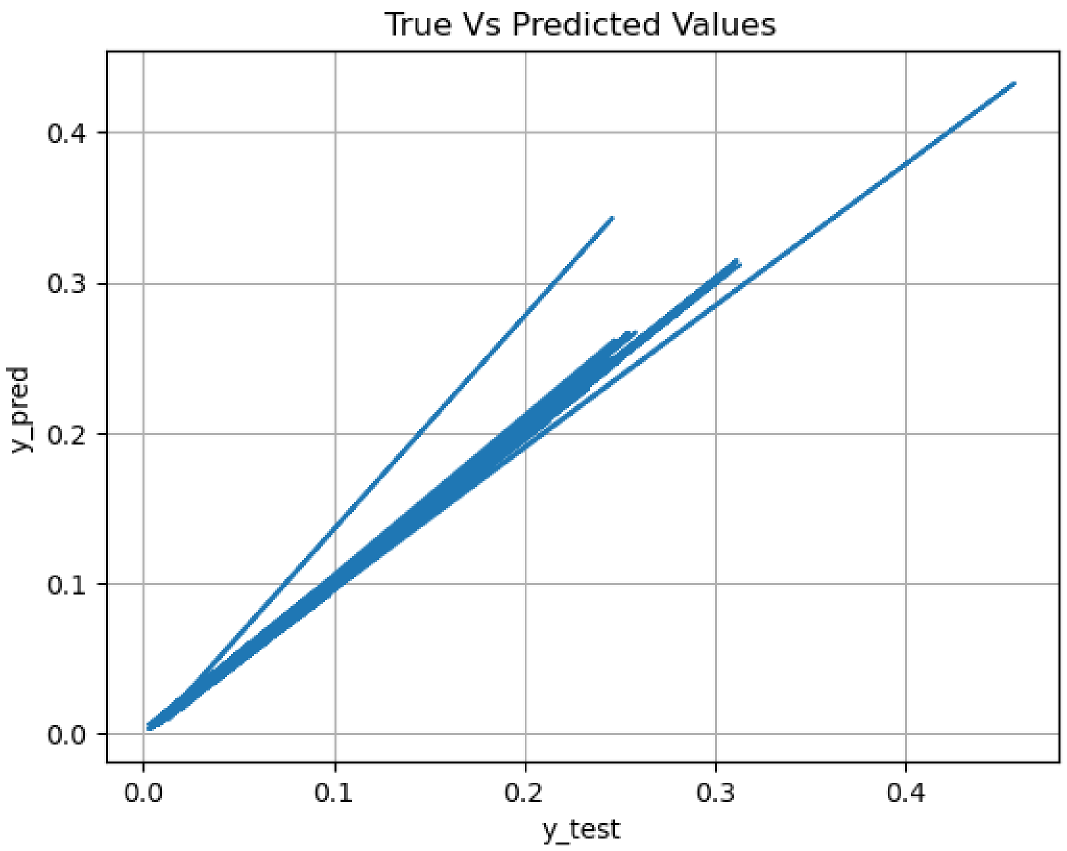


[39]:
```
print("Mean Squared Error:", mean_squared_error(y_test, rf_pred))
print("RMSE:",np.sqrt(mean_squared_error(y_test,rf_pred)))
print("R^2 Score:", r2_score(y_test, rf_pred))
```

```
Mean Squared Error: 2.8120224572065076e-07
RMSE: 0.0005302850608122491
R^2 Score: 0.9997397636137041
```

14.3 Plot of True values vs Predicted values

[40]:
```
plt.plot(y_test,rf_pred)
plt.grid()
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.title("True Vs Predicted Values");
```



14.4 Remarks :

- It is a Underfit model a scattered pattern with less clear trend or relationship, it suggests that the model is too simple (underfit). It fails to capture the underlying structure of the data.
- Due to less clear trend , model only knows few patterns , thus the histogram is a bit narrow.

15 Rohan Mekala , 22B2106

16 4. Adaboost

[41]:
```
from sklearn.ensemble import AdaBoostRegressor
```

[42]:
```
adaBoost_model = AdaBoostRegressor()
```

16.1 Training model on train data

[43]:
```
adaBoost_model.fit(X_train,y_train)
```

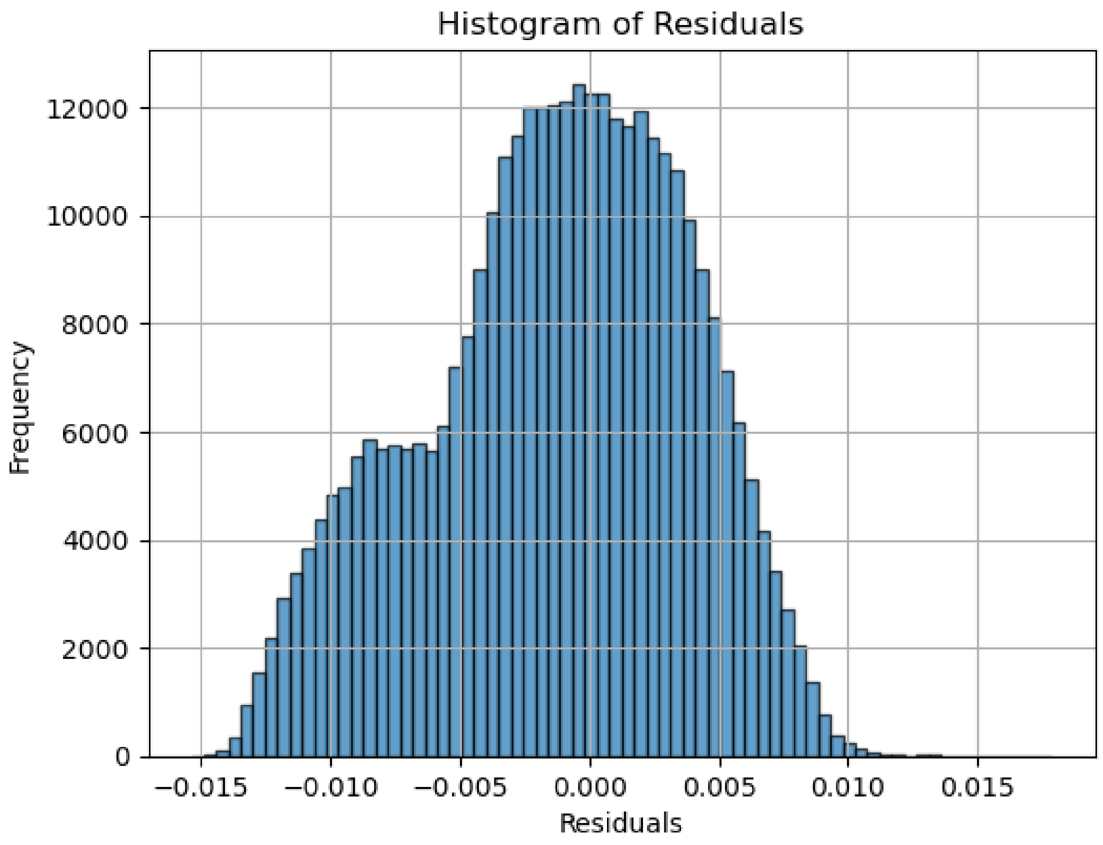
[43]:
```
AdaBoostRegressor()
```

16.2 Predictions on test data

[44]:
```
ada_pred = adaBoost_model.predict(X_test)
```

[45]:
```
ada_residuals = y_test-ada_pred
```

[46]:
```
plt.hist(ada_residuals, bins=70, edgecolor='black', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



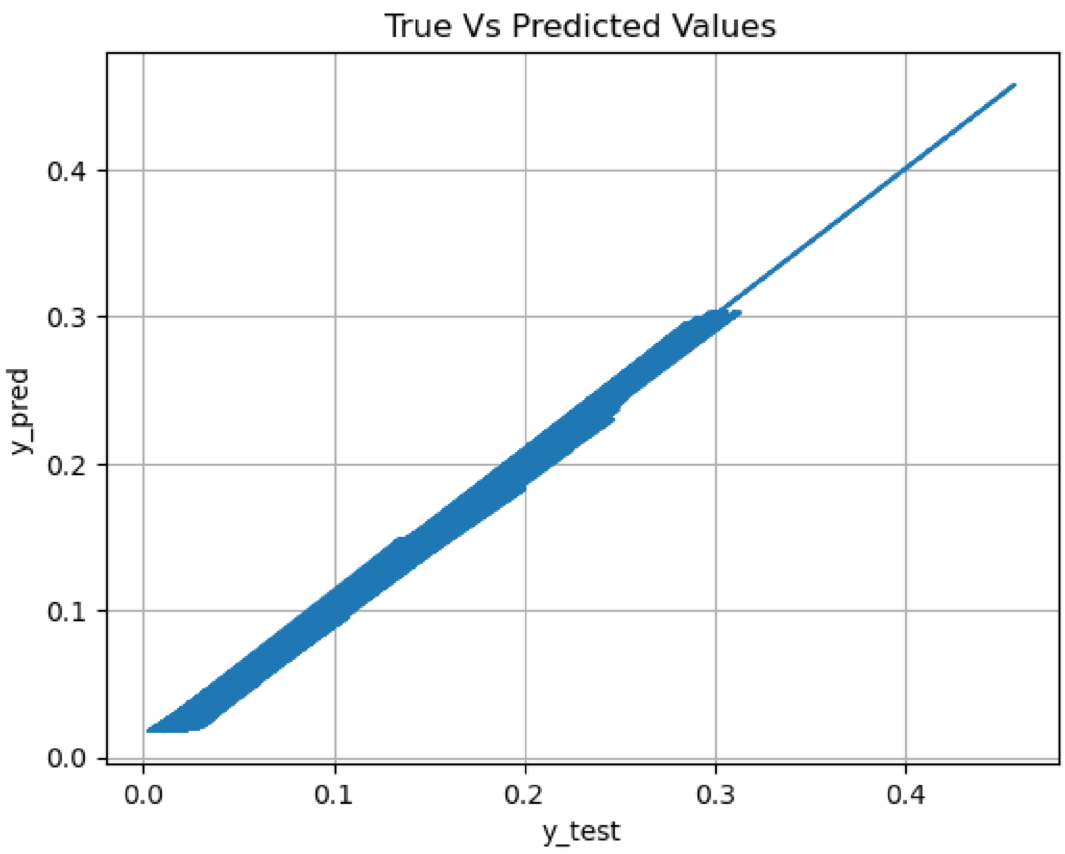
16.3 Metric evaluations

[47]:
```
print("Mean Squared Error:", mean_squared_error(y_test, ada_pred))
print("RMSE:",np.sqrt(mean_squared_error(y_test,ada_pred)))
print("R^2 Score:", r2_score(y_test, ada_pred))
```

```
Mean Squared Error: 2.707754419833397e-05
RMSE: 0.0052036087668405374
R^2 Score: 0.9749413016461295
```

16.4 Plot of True values vs Predicted values

[48]:
```
plt.plot(y_test,ada_pred)
plt.grid()
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.title("True Vs Predicted Values");
```



16.5 Remarks:

- It is a good model, as the points to cluster closely around the 45-degree line, indicating a strong linear relationship between actual and predicted values.
- A thick cluster in 45 deg , also can be seen as due to residuals are normally but has a wider plot.

17 5. Support Vector Regression

[1]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
```

18 Standardizing the features

[29]:
```
scaler = StandardScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)
```

[30]:
```
# Define SVR model
svr_model = SVR()
```

[31]:
```
# Define parameter grid for grid search (simplified)
param_grid = {
    'C': [0.01, 0.1],
    'kernel': ['rbf'],
}
```

19 Perform grid search with cross-validation

[36]:
```
grid_model = GridSearchCV(svr_model, param_grid=param_grid, cv=5, n_jobs=-1)
grid_model.fit(scaled_X_train, y_train)
```

[36]:
```
GridSearchCV(cv=5, estimator=SVR(), n_jobs=-1,
             param_grid={'C': [0.01, 0.1], 'kernel': ['rbf']})
```

[37]:
```
# Get best parameters
best_params = grid_model.best_params_
```

20 Preditions on test data

[40]:
```
grid_preds = grid_model.predict(scaled_X_test)
```

21 Metric Evaluations

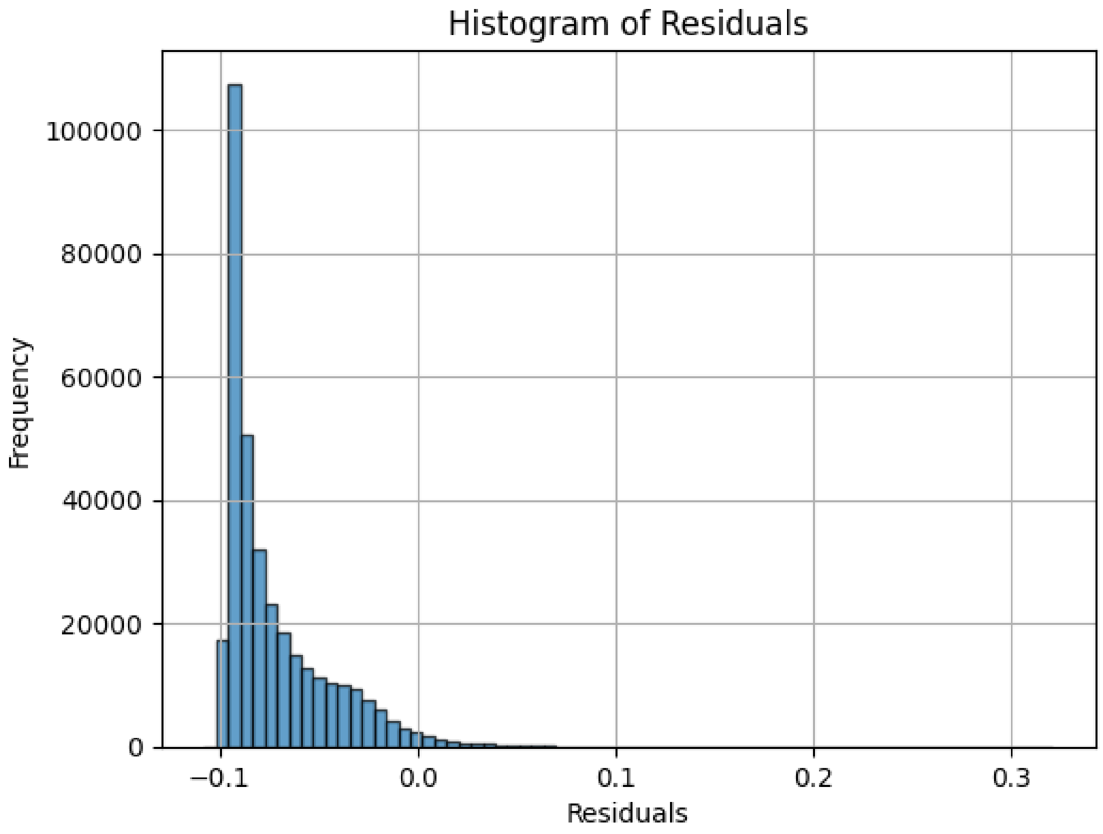
[42]:
```
mse = mean_squared_error(y_test, grid_preds)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, grid_preds)

print("SVR Model Evaluation:")
print("Best Parameters:", best_params)
print("Mean Squared Error:", mse)
print("RMSE:", rmse)
print("R^2 Score:", r2)
```

```
SVR Model Evaluation:
Best Parameters: {'C': 0.01, 'kernel': 'rbf'}
```

```
Mean Squared Error: 0.005952039588127247
RMSE: 0.07714946265611476
R^2 Score: -4.5082677932927755
```

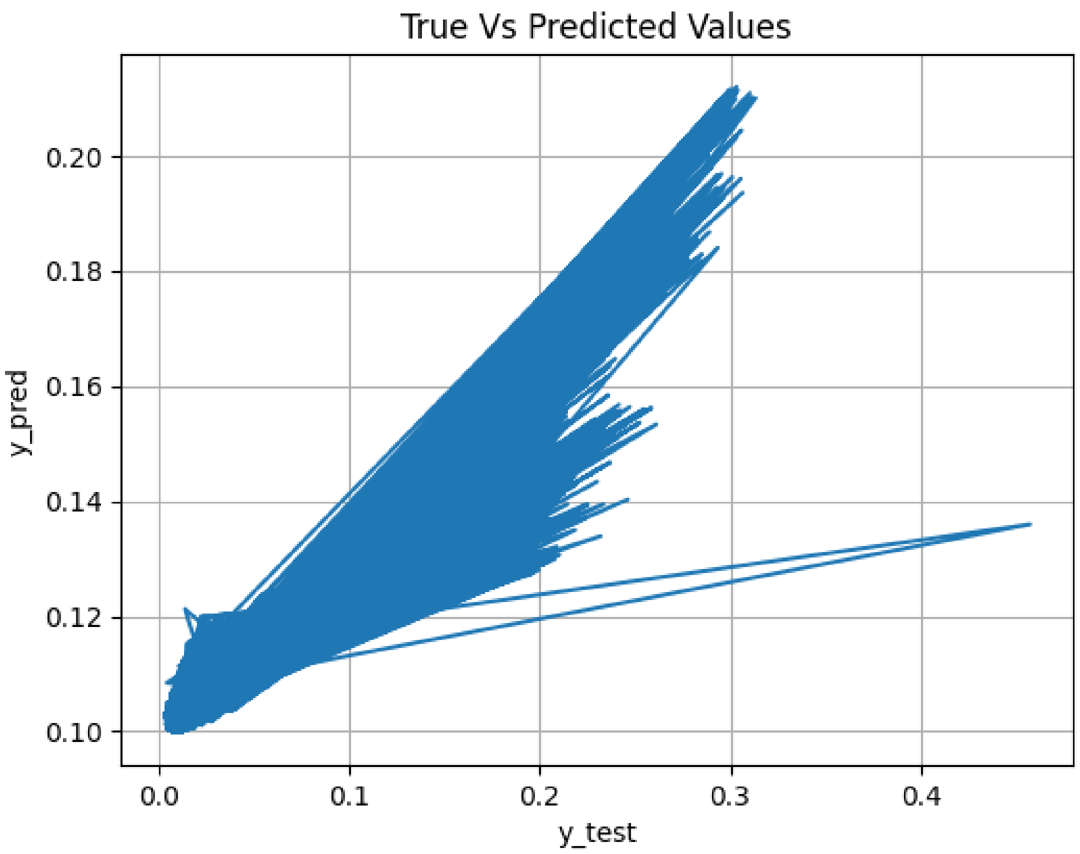
[45]:
```
grid_residuals = y_test - grid_preds
```

[46]:
```
plt.hist(grid_residuals, bins=70, edgecolor='black', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



22 Plot of True values vs Predicted values

[47]:
```
plt.plot(y_test,grid_preds)
plt.grid()
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.title("True Vs Predicted Values");
```



23 Remarks :

- The residual histogram appears right-skewed, it indicates that the residuals, or the differences between observed and predicted values in a regression analysis, are not evenly distributed around zero.
- It suggests that the model tends to underestimate the dependent variable in certain cases.
- From true vs predicted value plot , it doesn't flow in 45 degrees. It is clustered in other directions , indicating Overfitting of the model.
- This overfitting and no-normal distributions has led to abnormal R^2.

24 6. XGBoost

[48]:
```
import xgboost as xgb
```

[49]:
```
xgb_reg = xgb.XGBRegressor()
```

25 Model Training

[50]:
```
xgb_reg.fit(scaled_X_train, y_train)
```

[50]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, …)
```

26 Predictions on test data

[51]:
```
xgb_preds = xgb_reg.predict(scaled_X_test)
```

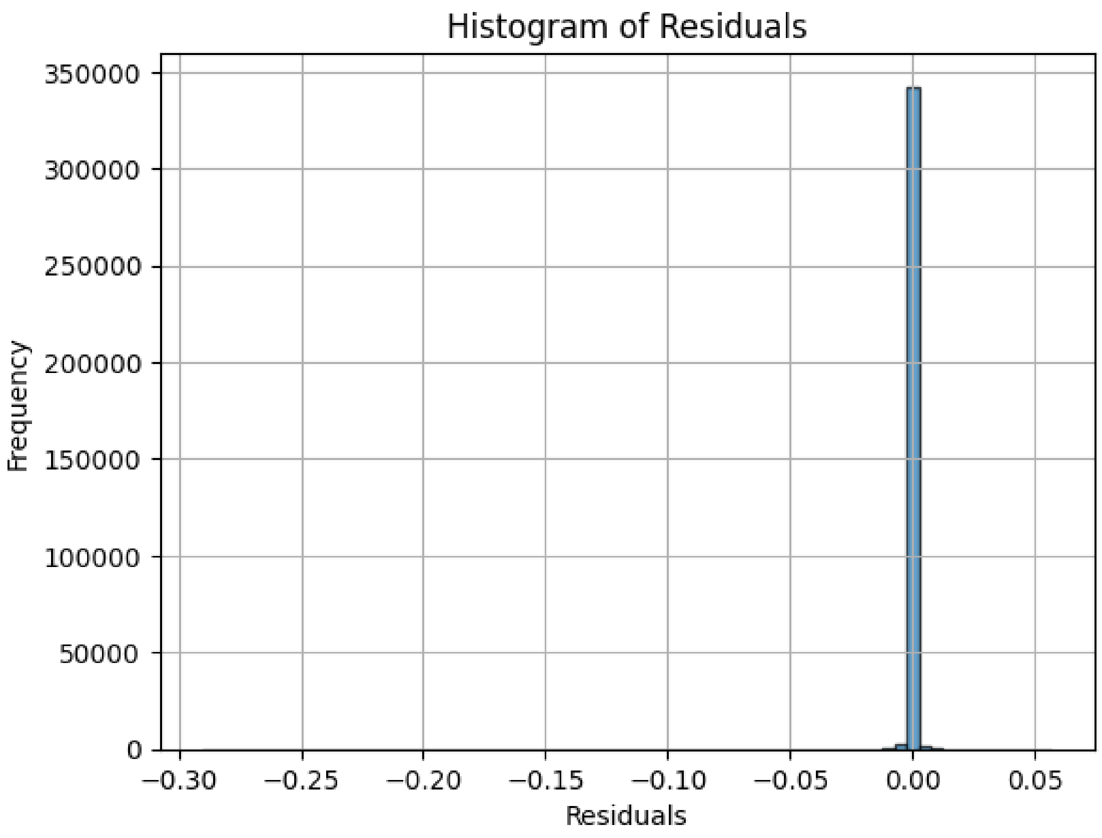
27 Metric Evaluations

[52]:
```
xgb_mse = mean_squared_error(y_test, xgb_preds)
xgb_rmse = np.sqrt(xgb_mse)
xgb_r2 = r2_score(y_test, xgb_preds)

print("\nXGBoost Model Evaluation:")
print("Mean Squared Error:", xgb_mse)
print("RMSE:", xgb_rmse)
print("R^2 Score:", xgb_r2)
```

```
XGBoost Model Evaluation:
Mean Squared Error: 1.5297256617060635e-06
RMSE: 0.0012368207880311777
R^2 Score: 0.9985843275283725
```

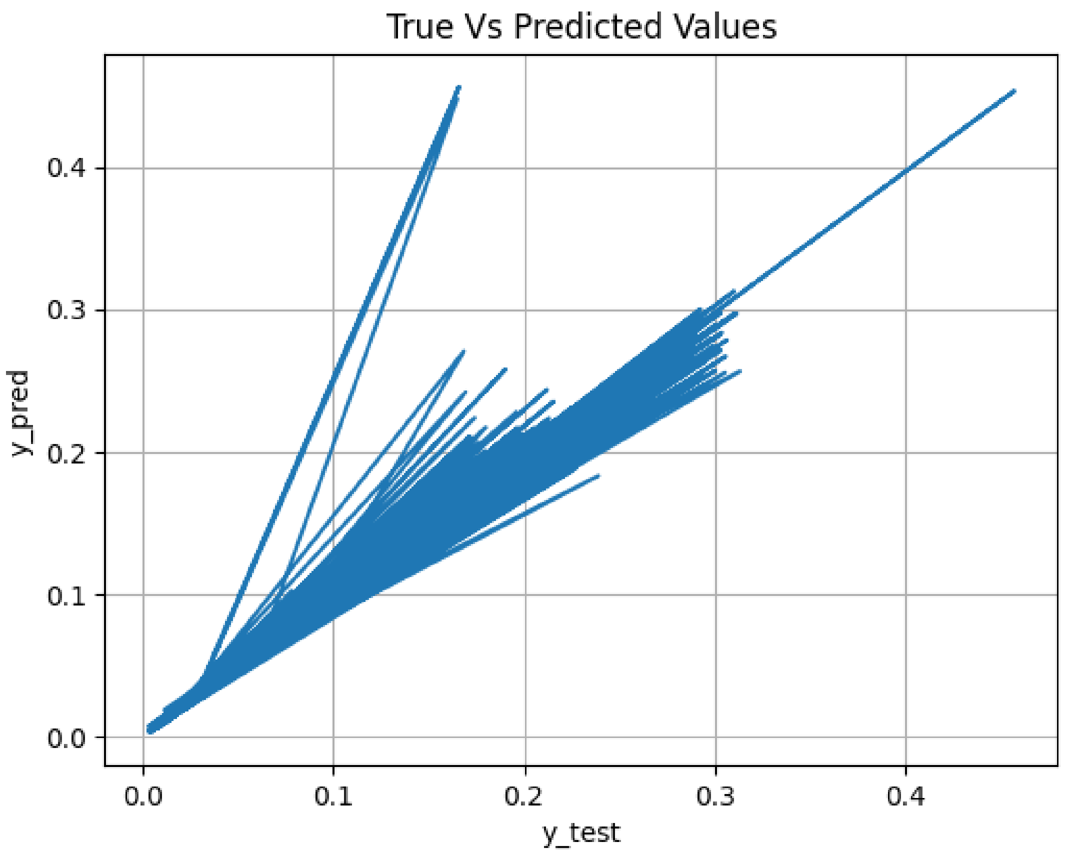
[53]:
```
xgb_residuals = y_test - xgb_preds
```

[54]:
```
plt.hist(xgb_residuals, bins=70, edgecolor='black', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



28 Plot of True values vs Predicted values

[55]:
```
plt.plot(y_test,xgb_preds)
plt.grid()
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.title("True Vs Predicted Values");
```



29 Remarks :

- The residual is normally distributed but for very small no of bins .
- The true vs predicted values has some points along 45 degrees , but rest are clustered to other directions.
- Due to normal and some points along 45 degree , we can consider it as a better model than SVR but in overall ,it is overfitted .

---

30 Priyanshi , 22B2213

31 7. Grad boosting

[49]:
```python
from sklearn.ensemble import GradientBoostingRegressor
```

[50]:
```python
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,␣
↪max_depth=3)
```

31.1 Model Training

[51]:
```
gb_model.fit(X_train,y_train)
```

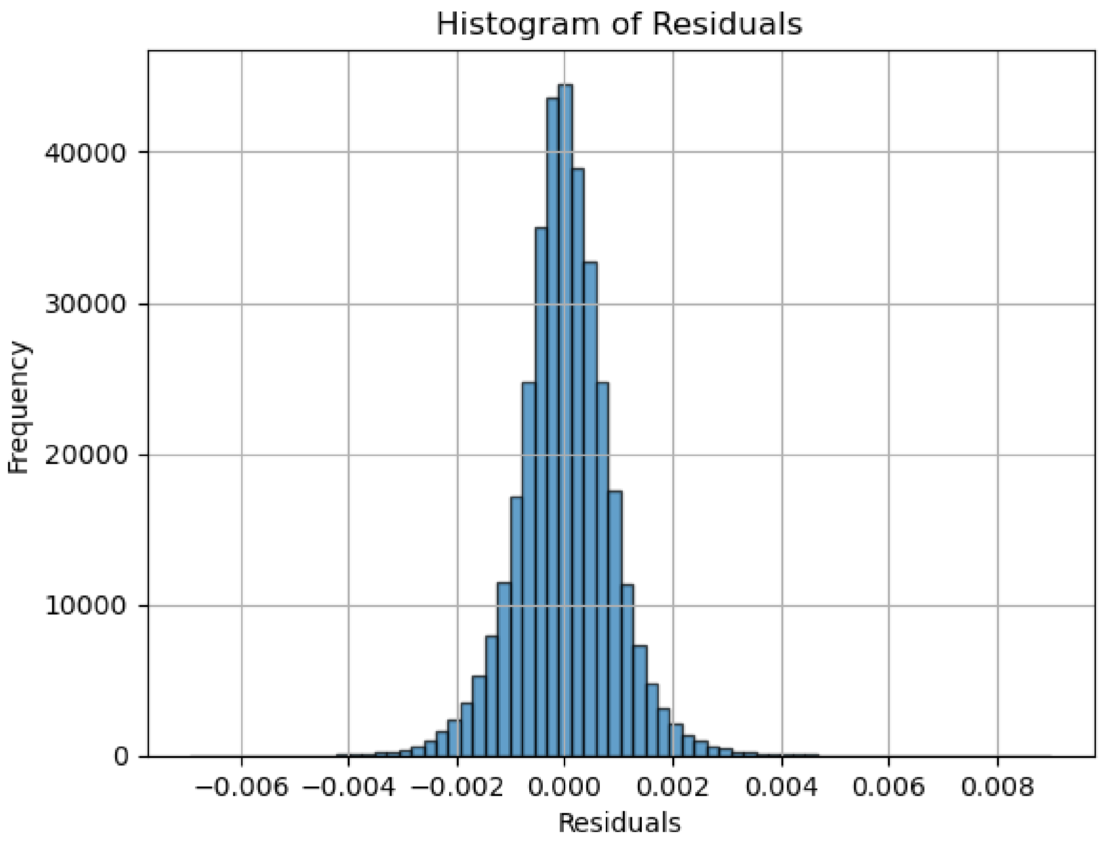
[51]:
```
GradientBoostingRegressor()
```

31.2 Predictions on test data

[52]:
```
gb_pred = gb_model.predict(X_test)
```

[53]:
```
gb_residuals = y_test - gb_pred
```

[54]:
```
plt.hist(gb_residuals, bins=70, edgecolor='black', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



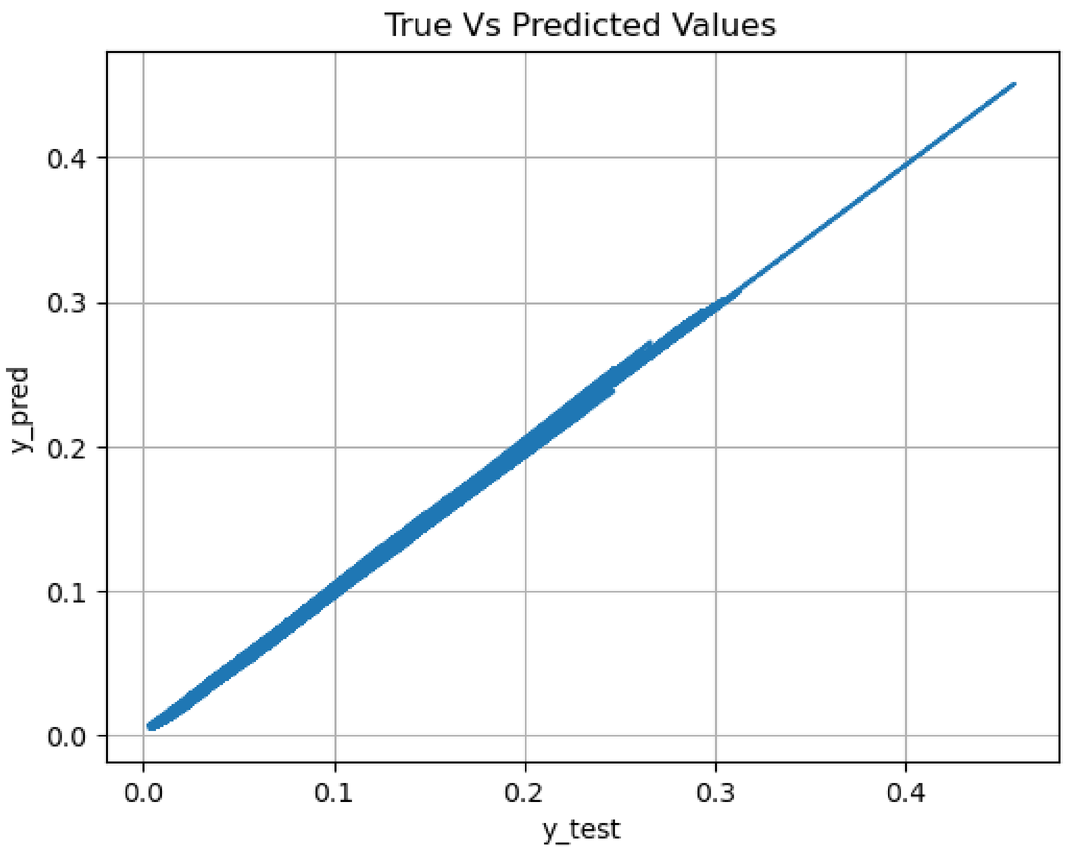
31.3 Metric Evaluations

[55]:
```
print("Mean Squared Error:", mean_squared_error(y_test, gb_pred))
print("RMSE:",np.sqrt(mean_squared_error(y_test,gb_pred)))
print("R^2 Score:", r2_score(y_test, gb_pred))
```

```
Mean Squared Error: 7.484544869115448e-07
RMSE: 0.0008651326412241909
R^2 Score: 0.9993073487358478
```

31.4 Plot of True values vs Predicted values

[56]:
```
plt.plot(y_test,gb_pred)
plt.grid()
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.title("True Vs Predicted Values");
```



31.5 Remarks :

- It is a good model,as the points to cluster closely around the 45-degree line, indicating a strong linear relationship between actual and predicted values.
- The residuals are very nicely normally distributed which means that the model adequately captures the relationship between the variables

32 8. Nearest Neighbor Regression

[57]:
```python
from sklearn.neighbors import KNeighborsRegressor
```

[58]:
```python
knn_regressor = KNeighborsRegressor()
```

32.1 Model Training

[59]:
```python
# Train the model
knn_regressor.fit(scaled_xTrain, y_train)
```

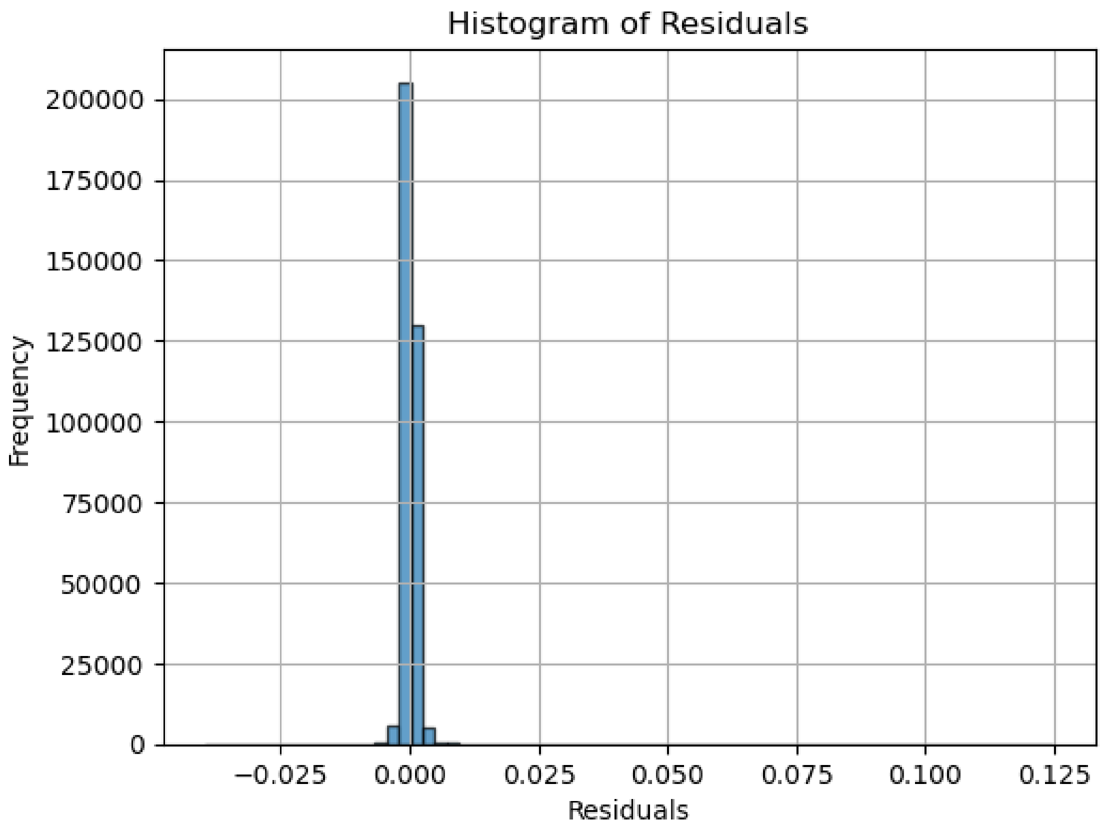
[59]:
```
KNeighborsRegressor()
```

32.2 Predictions on test data

[60]:
```python
knn_pred = knn_regressor.predict(scaled_xTest)
```

[61]:
```python
knn_residuals = y_test-knn_pred
```

[62]:
```python
plt.hist(knn_residuals, bins=70, edgecolor='black', alpha=0.7)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

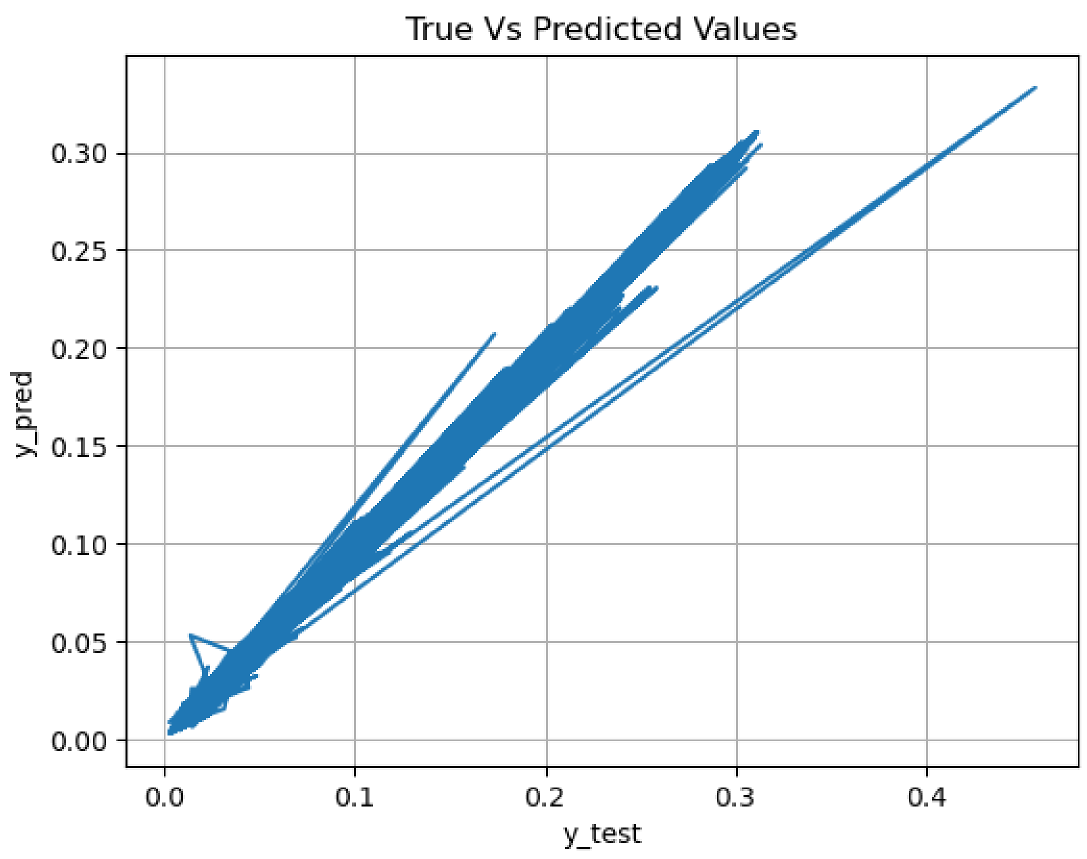


32.3 Metric evaluations

[63]:
```
print("Mean Squared Error:", mean_squared_error(y_test, knn_pred))
print("RMSE:",np.sqrt(mean_squared_error(y_test,knn_pred)))
print("R^2 Score:", r2_score(y_test, knn_pred))
```

```
Mean Squared Error: 1.1289818029806335e-06
RMSE: 0.0010625355537489715
R^2 Score: 0.9989551927515777
```

[64]:
```
plt.plot(y_test,knn_pred)
plt.grid()
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.title("True Vs Predicted Values");
```



32.4 Remarks:

- The plot shows a tight cluster of points but not along the 45-degree line, it suggests that the model is too complex (overfit). It may be fitting to noise rather than the true underlying relationship.
- The histogram is also not a proper normal distribution plot.

33 Creating a Dataframe to show the different metrics

[65]:
```
import pandas as pd

model_names = ['Linear Reg', 'Elastic Net', 'Ada Boost', 'Gradient Boost',␣
 ↪'Nearest Neighbors', 'SVR', 'Random Forest', 'XG Boost']
mse_values = [mean_squared_error(y_test, predictions),␣
 ↪mean_squared_error(y_test, test_predictions), mean_squared_error(y_test,␣
 ↪ada_pred), mean_squared_error(y_test, gb_pred), mean_squared_error(y_test,␣
 ↪knn_pred), 0.005952039588127247,mean_squared_error(y_test, rf_pred), 7.
 ↪484544869115376e-07]
```

```
rmse_values = [np.sqrt(mean_squared_error(y_test, predictions)), np.
 ↪sqrt(mean_squared_error(y_test, test_predictions)), np.
 ↪sqrt(mean_squared_error(y_test, ada_pred)), np.
 ↪sqrt(mean_squared_error(y_test, gb_pred)), np.
 ↪sqrt(mean_squared_error(y_test, knn_pred)), 0.077149,np.
 ↪sqrt(mean_squared_error(y_test, rf_pred)), 0.001236]
r2_values = [r2_score(y_test, predictions), r2_score(y_test, test_predictions),␣
 ↪r2_score(y_test, ada_pred), r2_score(y_test, gb_pred), r2_score(y_test,␣
 ↪knn_pred), -4.508267, r2_score(y_test, rf_pred),  0.998584]



df_metric = pd.DataFrame({'Model': model_names, 'MSE': mse_values, 'RMSE':␣
 ↪rmse_values, 'R^2': r2_values})

df_metric
```

[65]:
```
              Model           MSE      RMSE       R^2
0        Linear Reg  2.063141e-06  0.001436  0.998091
1       Elastic Net  2.296623e-06  0.001515  0.997875
2         Ada Boost  2.707754e-05  0.005204  0.974941
3    Gradient Boost  7.484545e-07  0.000865  0.999307
4 Nearest Neighbors  1.128982e-06  0.001063  0.998955
5               SVR  5.952040e-03  0.077149 -4.508267
6     Random Forest  2.812022e-07  0.000530  0.999740
7          XG Boost  7.484545e-07  0.001236  0.998584
```

34 Thank You

34.1 - Group 10

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

```
[ ]:
```