Housing Prices

Quid Zohar Morbiwala

Submitted for the Degree of Master of Science in

Data Science and Analytics

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

February 5, 2025

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 13471

Student Name: Quid Zohar Morbiwala

Date of Submission: February 5, 2025

Signature: 

Abstract

The objective of this project is to predict house prices by applying various machine learning algorithms to explore different housing datasets and comparing their performance. This includes regression models which include **K-Nearest Neighbors, Random Forest, Kernelized Ridge Regression, and Neural Networks** which are implemented using Python. The algorithms were selected for assessment and in the intent was to compare their relative performance at this task These models are applied to the **Boston Housing Dataset, the California Housing Dataset, Ames Housing and United Kingdom Dataset**. The performance of each model is visualized and compared. The report details the background knowledge required to implement these models, the pre processing steps for each data set, and an analysis of the results. Hyper-parameters for the algorithms used were optimally selected and the cleaned data set was split using five-fold cross-validation to reduce the risk of bias. An optimal subset of hyper parameters for the two algorithms was selected through the grid search algorithm for the best prediction. The **Random Forest** was found to consistently perform better than the **kNN algorithm** in terms of smaller errors and be better suited as a prediction model for the house price problem.

Contents

1 Introduction and background

The problem of searching for patterns in data is a fundamental one and has a long and successful history [3]. Accurately evaluating the value of real estate is a critical issue for many business stakeholders, including homeowners, home purchasers, real estate brokers, creditors, investors and it's also a challenge from a investment point of view. Critically analysing the house and predicting it's value to minimize the risk involved is important for a real estate business to occur. Though it is general known that pricing is affected by criteria such as size, number of rooms, and location, there are many others. Houses have a variety of features that may not have a comparable value owing to their location. For example, we assume that a large house could sell for a greater price if it is located in a desired wealthy people community rather than in a poor neighborhood.

There are other factors at work. Furthermore, prices are subject to variations in market demand as well as the unique characteristics of each case, such as when a property must be sold quickly. A property's sale price can be forecasted in a variety of methods, but regression approaches are frequently used. In essence, all regression approaches include one or more predictor variables as input and a single target variable as output. In this research, we compare several machine learning methods that tries to learn to estimating the selling price of properties based on a variety of characteristics such as the area, the number of bedrooms and bathrooms, and the geographical location. Additionally, applied modelling and itś application has been explained in section A

1.1 Research Questions

Rising interest rates are the main reason behind a predicted slow down in the housing market in 2023 and 2024. The study answers the following some fundamental questions like

- Research question 1: What are the factors that have affected house prices in Boston, Ames, California, United Kingdom over the years?
- Research question 2: Which machine learning algorithm performs better and has the most accurate result in house price prediction? and explore why is performs better?

1.2 Regression Problem

Regression problems start with a collection of potential predictors. Some of these may be continuous measurements, like the height or weight of an object. Some may be discrete but ordered, like a doctor's rating of overall health of a patient on a nine-point scale. Other potential predictors can be categorical, like eye color or an indicator of whether a particular unit received a treatment [15]. In regression problems, an attractive procedure is to fill in the missing values by fitting regression models. For example, to impute missing values for a particular predictor X1 based on a set of other predictors X2, one could build a regression model for E(X1—X2) based on complete data. The fitted model can be used to estimate a predicted value for X1 based on X2 for the cases for which X1 is unobserved. In general

this can give fill in values that are “too good” in the sense that the imputed values will be less variable than would the unobserved “true” values [15].

1.3 Limitations

The local data will be from open sources. [8] [11] [13] This project contains a list of features, that matches the public data set’s features. There is no guarantee that the data will be available in time nor contains the exact list of features. The focus on the specific chosen algorithm, starting from the regression techniques to the advanced one. Likewise,the artificial neural network that has many techniques and a wide area and several training methods that do not fit in this study.

1.4 Thesis structure

The thesis follows structure as: Section 1 introduces the study and provides some background research. Section 2 shows the ml framework followed in the project. Section 3 gives an overview of the algorithms used in this thesis. Section 3 describes the methodology used in this research, as well as the different experiment designs. Section 4 includes the methodologies employed in the research, analysis as well as the theoretical findings. Section 5 depicts the experimental results and model comparisons. Section 6 shows the pipe-lining and conformal Validity for pipeline. Finally, Section 7 finishes conclusion for further work. Section 8 self assesment. Section 9 shows Professional issues.

2 Machine learning framework used in this research

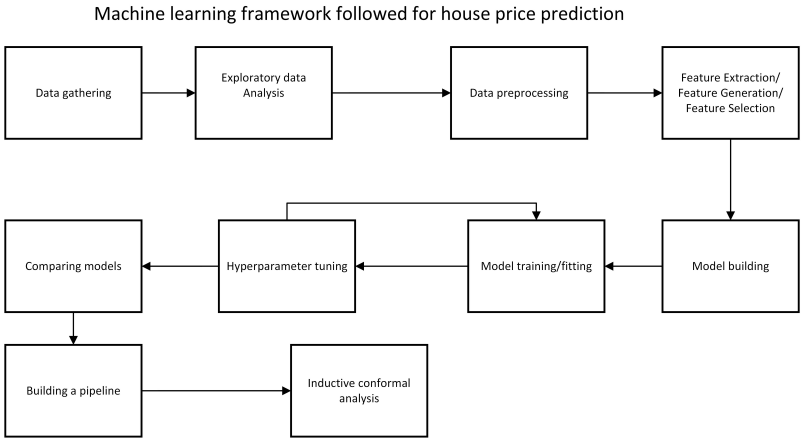


Figure 1: Machine learning Framework

2.1 Data Collection

2.1.1 Data gathering

The process of collecting data for each of the four house price prediction data-sets from various sources online. Boston, Ames, California, and United Kingdom.

2.1.2 Exploratory data analysis

An initial exploration of the datasets to understand the distribution of variables, identify potential issues, and gain insights into the data's structure. Summary statistics, data visualization, and initial observations of the data. **Data processing** This stage involves cleansing the data and ensuring that it contains meaningful insights with minimal junk so it can be used for further analysis.
Checking for duplicates A crucial step in ensuring the data's integrity by detecting and handling any duplicate records that might exist within each data-set. In this project we are keeping the first values and removing all other duplicates.
Removing missing values Addressing missing data points within the data-sets. Explain the techniques used for identifying missing values and the strategies applied to handle them effectively.
Removal of outliers Detecting and dealing with outliers, a critical step to improve the robustness of the predictive models. Describe the use of Z-score values to identify outliers and the decision criteria for their removal. **Scaling up features** In this research we have MinMax scaling to normalize the features of the data-sets. Explain the rationale behind this scaling technique and its impact on the model-building process.

2.2 Feature engineering

The creation of new features enhance the predictive power of the models. These include Log of features or target most relevant for house price, Creating polynomial features by raising existing features to higher powers like squared features, Combining features example creating a total rooms by adding number of bedrooms and bathrooms, Converting categorical variables by binning or discretizing their values, Converting date/time features, such as day of the week, month, or time of day of years and months, Making geographical information features example distance from sea, population density type, Creating dummy values from categorical values.

2.2.1 Correlation analysis

Exploring the correlations between features in each data-set. Identify highly correlated variables and their potential impact on the model's performance.

2.2.2 Feature selection

A careful selection process to choose the most relevant features for building accurate and efficient predictive models. These include removing Unique values, removing uncorrelated values to target, removing Single value record.

2.3 Training and test splitting

Identifying the target variable (house prices) and splitting the data into training and testing sets. Explain the importance of this step in evaluating the model's performance. This involves random sampling the data so that there is no bias within the data.In Python's scikitlearn library, the train_test_split function helps us to shuffle the data-set and split it into training and test sets. By default, it extracts 75% of the data into the training set and 25% into the test set. We can modify this split proportion by changing the parameters of the function but in general, a test set containing 25% of the data is considered a good rule of thumb.

2.4 Building the model and training the data

Implementing and applying models from their mathematical representation these include these four algorithms that are K-Nearest Neighbors (KNN), Random Forest, Kernelized Ridge Regression and Multilayer Perceptron (MLP). Fitting them and making sure they are fitted properly to scaled test and train data sets.This involves a long computational time for training as these data sets involve complex computations done for each sets.

2.5 Hyper-parameter tuning

Fine-tuning the model by exploring different hyper-parameter configurations. This involves building parameter grids and exploring by using looping or Gridsearch within these parameters. These have a high impact on the results as incorrect grids can lead to sub-optimal results and we might have to rerun the hyper parameters tuning.

2.6 Model evaluation and comparing different algorithms

Evaluate the trained models using appropriate metrics, compare their performance, and identify the strengths and weaknesses of each algorithm for house price prediction.

2.7 Building a pipeline for the best algorithm

Based on the evaluation results, the best-performing algorithm is chosen. A pipeline is constructed, integrating data prepossessing, feature engineering, model training, and predictions.

2.8 Inductive conformal analysis for the best fitted model

An Inductive conformal predictor (ICP) provides validity to a model by producing prediction intervals that are guaranteed to have a certain level of coverage. An ideal ICP would have a coverage that is equal to or greater than 1 - ϵ for all significance levelsϵ, where significance is the probability that a new observation will fall outside the prediction interval. the average interval width represents the average size of the prediction intervals produced by the ICP. A smaller average interval width indicates that the ICP is producing more precise predictions, while a larger average interval width indicates that the predictions are less precise.

The average interval width is directly affected by the significance level used in the ICP. A lower significance level results in narrower prediction intervals and a smaller average interval width, while a higher significance level results in wider prediction intervals and a larger average interval width. By evaluating the average interval width as a function of the significance level, you can assess the trade-off between precision and coverage in the predictions made by the ICP on the built pipeline.

2.8.1 Nonconformity

Intuitively, a nonconformity measure is a way of measuring how different a new example is from old examples. Formally, a nonconformity measure is a measurable mapping $A : Z^{(*)} \times Z \to \overline{\mathbb{R}}$. To each possible bag of old examples and each possible new example, A assigns a numerical score indicating how different the new example is from the old ones and $Z^{(*)}$ simply represents the set of examples [6].

In a regression problem where examples are pairs of numbers, say $z_i = (x_i, y_i)$, we might define the nonconformity of a new example (x, y) as the absolute value of the difference between y and the predicted value $\hat{y}$ calculated from x and the old examples [6].

2.8.2 P-Values

Given a nonconformity measure A_n and a bag $\{z_1, \ldots, z_n\}$, we can compute the nonconformity score $\alpha_i = A_n(\{z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n\}, z_i)$ for each example z_i in the bag. The fraction $\frac{|\{j=1,\ldots,n:\alpha_j \geq \alpha_i\}|}{n}$ is called the p-value for z_i. It is the fraction of the examples in the bag as nonconforming as z_i. If it is small (close to its lower bound $\frac{1}{n}$ for a large n), then z_i is very nonconforming (an outlier). If it is large (close to its upper bound 1), then z_i is very conforming [6].

2.8.3 Conformal Predictors

Every nonconformity measure determines a confidence predictor. The conformal predictor determined by a nonconformity measure A_n is the confidence predictor Γ obtained by setting $\Gamma_\epsilon(x_1, y_1, \ldots, x_{n-1}, y_{n-1}, x_n)$ equal to the set of all labels $y \in Y$ such that

$$\frac{|\{i = 1, \ldots, n : \alpha_i \geq \alpha_n\}|}{n} > \epsilon$$

$\alpha_i = A_n(\{(x_1, y_1), \ldots, (x_{i-1}, y_{i-1}), (x_i, y), (x_{i+1}, y_{i+1}), \ldots, (x_{n-1}, y_{n-1}), (x_n, y)\}, (x_i, y_i))$
$\alpha_n = A_n(\{(x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}, (x_n, y))$

$$p(y) := \frac{\#i = n - m + 1, \ldots, n : \alpha_i \geq \alpha_y + 1}{m + 1}$$

in the calibration step $p(y)$ is the rank of (x^*, y) in the augmented calibration set divided by the size of the augmented calibration set.

$$\mathrm{k} = \lceil (1 - \epsilon)(m + 1) \rceil,$$

where $\lceil t \rceil$ (the ceiling of t) is the smallest integer $\geq t$. Set $c = \alpha(k)$. The prediction set for any test sample is $[\hat{y} - c, \hat{y} + c]$, where $\hat{y}$ is the point prediction for its label (computed from the training set proper).

3 Overview of Algorithms

3.1 Regression

Regression is a statistical method used to study the relationship between a dependent variable and one or more independent variables. A regression problem or regression predictive modelling is the task of approximating a mapping function **f** from input variables **X** to a continuous output. Linear regression, or ordinary least squares (OLS), is the simplest and most classic linear method for regression. Linear regression finds the parameters w and b that minimize the mean squared error between predictions and the true regression targets,y,on the training set [17]. Let there be an input vector $X^T = (X1, X2, ..., Xp)$. We want to predict a real-valued output Y. A Linear Regression model predicts the output as a linear function of the input output variable **y** [5]. In simple terms, it is a problem where we predict a continues value for a data point.

$$f : X \to y, y \in \mathbb{R}$$

$$y = \beta_0 + \beta_1 x + \epsilon$$

where y is the dependent variable (in this case, house price), x is the independent variable (a factor that may affect house price), β_0 is the y-intercept (the value of y when $x = 0$), β_1 is the slope of the regression line (the change in y for a unit change in x), ϵ is the error term (the difference between the observed value of y and the predicted value of y)

In multiple linear regression, where there are multiple independent variables, the formula becomes:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon$$

where, $x_1, x_2, ..., x_n$ are the independent variables and $\beta_1, \beta_2, ..., \beta_n$ are the coefficients for each independent variable.

3.1.1 Losses in regression

A loss function is a metric that measures how effectively a prediction model predicts the predicted outcome. The loss function in regression assesses how close or distant the predicted values are to the original label values. Loss functions are sometimes known as error functions as they provide information about the predicted error.

Residual Sum of Errors (RSS) The Residual Sum of Squares (RSS) is a metric used in regression that is used to find discrepancy between the observed values and the predicted values generated by a regression model. It represents the sum of the squared differences between the actual target values (y) and the predicted values ($f(x)$).The goal in regression is often to minimize the RSS, as a smaller RSS indicates a better fit of the model to the observed data.Mathematically, the formula for RSS is given by:

$$\text{RSS} = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

here, n represents the number of data points in the dataset, y_i is the observed target value for the i-th data point, and $f(x_i)$ is the predicted value generated by the regression model for the i-th data point.

Mean Squared Error (MSE) This loss function calculates the average of the squared differences between the predicted and true values. It is sensitive to outliers, as larger errors have a larger impact due to squaring. The formula of it is given by

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Mean Absolute Error (MAE) This loss function calculates the average of the absolute differences between the predicted and true values. It is less sensitive to outliers than MSE. Its formula is given by

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - f(x_i)|$$

Some additional losses are shown in section B.

3.1.2 Ridge Regression

Ridge regression is a regularization technique used in linear regression to mitigate the issues of multicollinearity and over-fitting. As we discussed above linear regression, which aims to find a linear relationship between predictors (features like number of bedrooms) and a target variable (house price). However, when dealing with datasets like California and Ames ect these contain highly correlated features, the coefficient estimates in linear regression can

become unstable and sensitive to small changes in the data. Ridge regression addresses this problem by introducing a penalty term that limits the magnitude of the coefficients. In ridge regression, the goal is to minimize the residual sum of squares (RSS) while also constraining the sum of the squared coefficients ($L2$-norm) to be smaller. There is an addition of the penalty term $\lambda \sum_{j=1}^{p} \beta_j^2$, which encourages the coefficients to be close to zero. This has the effect of "shrinking" the coefficients towards zero, thus reducing the impact of less important features. This regularization helps in dealing with multicollinearity and prevents over-fitting, making the model more stable and reliable.

$$\text{minimize} \quad RSS + \lambda \sum_{j=1}^{p} \text{minimize} \beta_j^2$$

this is an optimization problem where were are trying to minimize the RSS. RSS represent residual sum of squares, which measures the discrepancy between the predicted and actual target values.β_j represents the coefficient associated with the j-th feature.λ is the regularization parameter that controls the strength of the penalty. Here we are trying minimize RSS and beta such that the sum of both is minimum. A larger λ leads to smaller coefficient values, and a smaller λ allows coefficients to approach those of linear regression. The regularization parameter λ controls the balance between fitting the data and shrinking the coefficients. This technique is particularly useful when dealing with high-dimensional datasets with correlated features.

3.1.3 Why choose ridge regression?

Ridge Regression presents an effective alternative linear model aimed at addressing the issue of over-fitting. It not only seeks coefficients (β) that perform well on training data but also integrates an additional constraint to optimize the model. Ridge regression focuses on minimizing the coefficients magnitudes, steering them toward values close to zero. This nuanced approach aims to ensure that each feature's impact on the outcome is lessened while still achieving accurate predictions. To achieve this, Ridge Regression employs a regularization technique known as L2 regularization, which explicitly curbs model complexity to reduce over-fitting. In the context of Ridge Regression, the goal is to minimize a modified residual sum of squares:

$$RSS_{\text{ridge}}(\beta) = \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

where $\lambda \sum_{j=1}^{p} \beta_j^2$ introduces the shrinkage penalty, and λ serves as the regularization parameter.

The Ridge Regression equation above can be expressed in matrix form:

$$RSS_{\text{ridge}}(\beta) = (y - X\beta)^T (y - X\beta) + \lambda \beta^T \beta$$

solving for the coefficients β, the unique solution is obtained as:

$$\hat{\beta}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y$$

The prediction function is written as:

$$\hat{y} = X\hat{\beta}_{\text{ridge}} = X(X^T X + \lambda I)^{-1} X^T y$$

this formulation encapsulates the essence of Ridge Regression, allowing for controlled model complexity while maintaining accurate prediction capability. The regularization parameter λ plays a pivotal role in achieving this balance.

3.2 Kernelized Ridge Regression

3.2.1 Why use a Kernel function?

In the context of regression tasks, especially when dealing with intricate or nonlinear associations between predictor variables and the target variable, the integration of a kernel function introduces a potent technique for uncovering intricate data patterns. A kernel function facilitates the transformation of input variables into a higher-dimensional space called as Hilbert space, potentially revealing a more distinct structure or separability within the data. This transformation allows linear algorithms to effectively predict model complex nonlinear associations without the need for explicit expansion of the feature space [2].

3.2.2 Different Types of Kernels

The each kernel is tailored to capture specific data patterns and structures. Some commonly used kernels are:

Linear Kernel The linear kernel ($K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$) is a fundamental starting point. It assesses data relationships without any transformations, appropriate for linearly separable or straightforward problems.

$$K_{\text{linear}}(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$$

Polynomial Kernel The polynomial kernel ($K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + c)^d$) introduces polynomial transformations. It's beneficial for scenarios where data exhibits non-linearities but can still be separated through polynomial functions.where c is a constant and d is the degree of the polynomial.

$$K_{\text{polynomial}}(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + c)^d$$

Radial Basis Function (RBF) Kernel The RBF kernel ($K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)$) is versatile and excels at capturing intricate data relationships. It quantifies similarity between data points based on distance, and its smooth nature makes it adept at handling complex patterns.

$$K_{\text{RBF}}(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right)$$

where σ controls the width of the kernel. **Sigmoid Kernel** The sigmoid kernel ($K(\mathbf{x}, \mathbf{y}) = \tanh(\alpha \mathbf{x}^T \mathbf{y} + c)$) emulates neural networks' activation functions. It's useful for problems where we suspect the data relationships resemble sigmoid curves. where α and c are constants [2].

$$K_{\text{sigmoid}}(\mathbf{x}, \mathbf{y}) = \tanh(\alpha \mathbf{x}^T \mathbf{y} + c)$$

Note: In this project we have only used linear,polynomial,rbf kernels

3.2.3 Kernel Trick

The kernel trick allows us to implicitly map data into a higher-dimensional space without explicitly computing the transformation. Instead of transforming data and then performing computations in a high-dimensional space, the kernel trick directly operates in the original feature space but uses kernel evaluations to capture the transformed space's essence.

Mathematically, the kernel trick rewrites the inner products between data points in terms of the chosen kernel function, enabling SVMs and other algorithms to operate efficiently without explicit feature space expansions [2]. For instance, In the scenario of a house prices based on diverse attributes such as square feet footage, bedroomcount, and geographical location. While conventional linear regression may struggle to account for intricate interactions among these attributes, the application of a kernel function offers a promising avenue for capturing the underlying relationships.Incorporating geographical coordinates (latitude and longitude) as attributes. In the original feature space, a linear regression model might find it challenging to comprehend the geographic intricacies impacting house prices. However, by employing a kernel function the data can be projected into a higher-dimensional space where geographic patterns become more discernible. This enables the linear regression model to uncover complex relationships that might remain hidden within the original feature space.

3.2.4 Kernelized Ridge Regression Formulation

Kernelized Ridge Regression takes the familiar ridge regression formulation and combines it with the kernel trick.The loss function is calculated in terms of mean squared error and R^2 score. In its general terms, the aim of Kernelized Ridge Regression is to minimize the penalized residual sum of squares:

$$RSS_{\text{kernelized}}(\boldsymbol{\beta}) = \sum_{i=1}^{N} (y_i - \boldsymbol{\beta}^T \mathbf{h}(\mathbf{x}_i))^2 + \lambda \boldsymbol{\beta}^T \mathbf{K} \boldsymbol{\beta}$$

- y_i is the target variable for the i-th observation.
- $\mathbf{h}(\mathbf{x}_i)$ is the transformed feature vector using the kernel trick.
- $\mathbf{K}$ is the kernel matrix containing kernel evaluations between all pairs of observations.

- λ is the regularization parameter.

$$RSS_{\text{kernelized}}(\boldsymbol{\beta}) = (\mathbf{y}^T\mathbf{y}) - 2\boldsymbol{\beta}^T\mathbf{H}^T\mathbf{y} + \boldsymbol{\beta}^T\mathbf{H}^T\mathbf{H}\boldsymbol{\beta} + \lambda\boldsymbol{\beta}^T\mathbf{K}\boldsymbol{\beta}$$

Where, $\mathbf{y}$ is the vector of target values.$\boldsymbol{\alpha}$ is the vector of dual variables. $\mathbf{K}$ is the kernel matrix and λ is the regularization parameter.

3.2.5 Mathematical formulation

We aim to find the $\boldsymbol{\beta}$ that minimizes $RSS_{\text{kernelized}}$. To do this, we take the derivative of the objective function with respect to $\boldsymbol{\beta}$, set it to zero, and solve for $\boldsymbol{\beta}$:

$$\frac{\partial RSS_{\text{kernelized}}}{\partial \boldsymbol{\beta}} = -2\mathbf{H}^T\mathbf{y} + 2\mathbf{H}^T\mathbf{H}\boldsymbol{\beta} + 2\lambda\boldsymbol{\beta} = 0$$

Solving for $\boldsymbol{\beta}$:

$$\boldsymbol{\beta}_{\text{kernelized}} = (\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}^T\mathbf{y}$$

Once we've obtained the optimal coefficient vector $\boldsymbol{\beta}_{\text{kernelized}}$ using the derived formula, we can use it to predict the output (y_{pred}) for new input.

The predicted output y_{pred} is calculated as the dot product between the transformed input data and the optimal coefficient vector:

$$y_{\text{pred}} = \mathbf{h}(\mathbf{x}_{\text{new}})^T\boldsymbol{\beta}_{\text{kernelized}}$$

Here $\mathbf{x}_{\text{new}}$ is the new input data. $\mathbf{h}(\mathbf{x}_{\text{new}})$ is the transformed feature vector using the kernel trick and $\boldsymbol{\beta}_{\text{kernelized}}$ is the optimal coefficient vector obtained from the Kernelized Ridge Regression.

3.2.6 Hyper parameters for Kernelized Ridge Regression

Regularization Parameter (λ) The hyper-parameter λ (lambda) controls the trade-off between fitting the training data well and preventing overfitting. A higher value of λ increases the regularization strength, which can help prevent large coefficient values. The appropriate value of λ can be determined through techniques like cross-validation.

Kernel Function The choice of kernel function is a crucial hyperparameter in Kernelized Ridge Regression. Different kernel functions, such as linear, polynomial, radial basis function (RBF), and sigmoid, have varying effects on the model's ability to capture complex patterns. Selecting the right kernel function depends on the underlying data distribution and problem characteristics.The choice of kernel function plays a crucial role in Kernelized Ridge Regression. Different kernel functions capture distinct data patterns.

Kernel Parameters Some kernel functions, like the polynomial kernel, have additional parameters such as the degree of the polynomial. For the RBF kernel, there is the parameter γ that controls the width of the kernel's influence. The appropriate values of these parameters can significantly impact the performance of Kernelized Ridge Regression. Additional hyper parameters of Kernelized ridge regression are given in section D.2.

3.3 K-Nearest Neigbours

The knn algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset. To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset—its "nearest neighbors." [17] [10]

In Knn regression, the goal is to predict a continuous target variable for a given observation by looking at the values of its k-nearest neighbors. The basic idea is to identify the k observations in the training dataset that are closest to the given observation based on a distance metric (usually Euclidean distance). The predicted value for the target variable is then calculated as the average (or weighted average) of the target values of these k-nearest neighbors. Mathematically, for a new observation x_{new}, the predicted target value y_{pred} using KNN regression can be formulated as:

$$y_{\text{pred}} = \frac{1}{k} \sum_{i=1}^{k} y_i$$

Where, y_i is the target value of the i-th nearest neighbor. k is the number of neighbors to consider. The loss function is calculated in terms of mean squared error and R^2 score.

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

KNN regression does not assume any specific functional form for the relationship between predictors (features) and the target variable. Instead, it directly relies on the training data to make predictions. It is important to note that the choice of k influences the smoothness of the regression curve. Smaller values of k can result in a more flexible and potentially noisy regression curve, while larger values of k can lead to a smoother but potentially oversimplified regression curve.

3.3.1 Popular distance functions used in KNN

Minkowski distance It is a metric intended for real-valued vector spaces. The formula for Minkowski distance is given by:

$$D(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

where $\mathbf{x}$ and $\mathbf{y}$ are two vectors in an n-dimensional space, and p is a positive real number. where x and y are two vectors in an n-dimensional space, and p is a positive real number.

- When p=1, the Minkowski distance is equivalent to the Manhattan distance.
- When p=2, the Minkowski distance is equivalent to the Euclidean distance [10].

Manhattan distance This distance is also known as taxicab distance or city block distance. The distance between two points is the sum of the absolute differences of their Cartesian coordinates. The formula for Manhattan distance can be obtained by setting $p = 1$ in the Minkowski distance formula:

$$D(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

Euclidean distance This is the most widely used distance metric, as it is the default metric used by the most of the librarys. It is a measure of the true straight-line distance between two points in Euclidean space. The formula for Euclidean distance can be obtained by setting $p = 2$ in the Minkowski distance formula:

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

3.3.2 Weighted scheme KNN

In weighted k-nearest neighbors (KNN), a modification is made to the standard KNN algorithm by assigning specific weights to the labels of the nearest neighbors, instead of simply taking their mean. This modification allows the algorithm to give varying importance to different neighbors when making predictions. In this form, the k nearest neighbors of a test observation x are weighted according to their distance from x. The idea is that observations that are closer to x should impose a higher influence on decision making.

Mathematically, for a test observation x and its k nearest neighbors $x(i)$, the prediction y_{pred} using weighted KNN are formulated as:

$$y_{\text{pred}} = \frac{\sum_{i=1}^{k} w_i \cdot y(i)}{\sum_{i=1}^{k} w_i}$$

Where, w_i is the weight assigned to the i-th nearest neighbor, calculated based on its distance from x and $y(i)$ is the label of the i-th nearest neighbor.

The choice of weighting scheme can vary based on the specific problem and domain knowledge. For example, common weighting schemes include the inverse of the distance or

using a Gaussian function to assign weights. With this weighting scheme, the Weighted-kNN becomes:

$$\psi(x) = \begin{cases} 1, & \text{if} \quad \frac{1}{D}\sum_{i=1}^{k} \frac{1}{d[x(i),x]}\mathbb{I}\{y(i,x)=1\} > \frac{1}{D}\sum_{i=1}^{k} \frac{1}{d[x(i),x]}\mathbb{I}\{y(i,x)=0\} \\ 0, & \text{otherwise,} \end{cases} \quad [17]$$

3.3.3 Hyper parameters for KNN regressor

Number of Neighbors (k) The hyper-parameter k represents the number of nearest neighbors to consider when making predictions. A smaller value of k might lead to higher sensitivity to noise, while a larger k can lead to improved predictions but may not capture local patterns hidden.

Distance Metric The choice of distance metric to calculate the distance between data points is another important hyper-parameter. The choice of distance metric should align with the data characteristics.

Weighting Scheme kNN can be weighted based on the distance of neighbors from the test point. Two common weighting schemes are:

- **Uniform**: All neighbors are given equal weight.
- **Distance-based**: Neighbors closer to the test point have higher influence on predictions.

3.4 Random Forests Regression

3.4.1 Ensemble of Decision Trees

Random Forest leverages the concept of an ensemble, which involves combining the predictions of multiple individual models to create a stronger, more accurate model. In this case, the individual models are decision trees. Each decision tree in the ensemble is trained on a different subset of the data, promoting diversity and reducing the risk of overfitting.

3.4.2 Bagging

Bagging is a technique that stands for **Bootstrap Aggregating**. It aims to reduce the variance of individual models by training multiple models on bootstrapped subsets of the original data. In the context of Random Forests, each decision tree is trained on a different bootstrapped subset of the training data. This promotes diversity among the trees and prevents over-fitting by averaging their predictions during the ensemble phase [9].

3.4.3 Boosting

Boosting is an iterative technique that combines multiple weak learners (often simple models) to create a strong ensemble. In Random Forests, boosting involves training decision trees sequentially. Each subsequent tree is trained with increased emphasis on the samples that the previous trees misclassified. Boosting assigns weights to each data point, and each decision tree's importance is weighted based on its performance. The final prediction is a weighted combination of all trees' predictions. [9]

3.4.4 Decision trees

A decision tree is a hierarchical structure that makes decisions based on a sequence of features. It recursively splits the data into subsets based on feature values, aiming to classify or predict the target variable. However, a single decision tree might be prone to over-fitting and lack generalization power.

3.4.5 How Random Forest construction

The Random Forest algorithm constructs an ensemble of decision trees as shown in below steps. You can see it structure from section D.

1. A random subset of the training data is selected for each decision tree.
2. For each tree, a random subset of features is chosen as potential split candidates at each node.
3. The decision tree is grown using these random subsets, resulting in a diverse set of trees.
4. During prediction, each tree in the ensemble provides its prediction, and the final prediction is determined by averaging (regression) or majority vote (classification).

3.4.6 Random Forest Formula and Loss function

A regression tree divides the predictor space X into J distinct and non-overlapping regions: $R_1, R_2, \ldots, R_J$.

For a new test object x_0 and a trained regression tree

$$\hat{y} = \frac{1}{|R_{J_0}|} \sum_{n:x_n \in R_{J_0}} y_n$$

The loss function of a regression tree is given by

$$MSE(R_j) = \frac{1}{|R_j|} \sum_{n:x_n \in R_j} (y_n - \hat{y}_{R_j})^2$$

The hyper parameters used in Random foresr are explained in section D.1.

3.5 MultiLayered Perceptron

MultiLayer Perceptron (MLP) is a fundamental artificial neural network architecture widely used for various machine learning tasks, including classification and regression. It consists of multiple layers of interconnected neurons, each layer contributing to feature extraction and transformation. In figure 2, we can see a visualization of the mlp architecture [12].

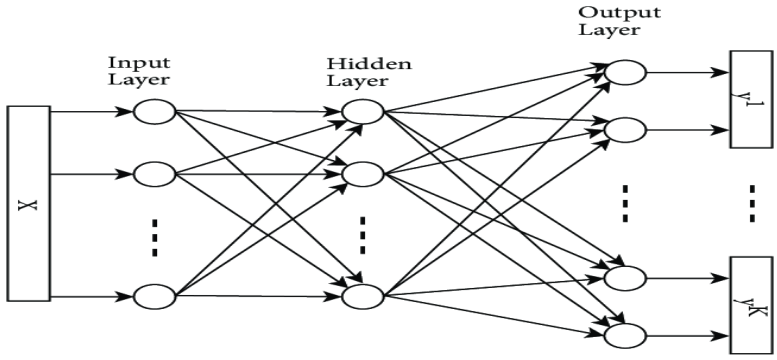


Figure 2: MultiLayer Perceptron Architecture

3.5.1 Architecture

An MLP typically comprises three types of layers:

- **Input Layer**: Receives the input features.
- **Hidden Layers**: Intermediate layers consisting of neurons that apply linear transformations followed by activation functions.
- **Output Layer**: Generates the final predictions.

The connections between neurons have associated weights that are adjusted during training.

3.5.2 Activation Functions

Activation functions introduce non-linearity and enable MLPs to capture complex relationships in data. Common activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Tanh.

Sigmoid Activation Function The sigmoid activation function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

It maps input values to the range (0, 1), making it suitable for binary classification problems. However, it suffers from vanishing gradients and might cause slower convergence during training.

ReLU (Rectified Linear Unit) The ReLU activation function is defined as:

$$\text{ReLU}(x) = \max(0, x)$$

It replaces negative values with zero and keeps positive values unchanged. ReLU is computationally efficient and helps mitigate the vanishing gradient problem, making it widely used in deep networks.

Hyperbolic Tangent (Tanh) The hyperbolic tangent activation function is defined as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

It maps input values to the range (-1, 1). Tanh is zero-centered and can help mitigate the vanishing gradient problem compared to sigmoid.

Some additional activation functions and regression functions are give in section C

3.5.3 Regression Formula

In the context of regression, an MLP's output layer typically has a single neuron with a linear activation function. The regression formula for MLP can be expressed as:

$$y_{\text{pred}} = w \cdot x + b$$

where w is the weight vector and b is the bias.

3.5.4 Loss Function

The loss function measures the difference between predicted and actual values. In regression tasks, Mean Squared Error (MSE) is commonly used as the loss function:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - y_{\text{pred},i})^2$$

3.5.5 Hyperparameters

- **Number of Hidden Layers and Neurons**: The architecture of hidden layers, including the number of layers and neurons in each, affects the model's capacity to learn complex patterns.
- **Activation Function**: The choice of activation function determines the type of non-linearity introduced in the network.
- **Learning Rate**: Controls the step size during weight updates and impacts the convergence speed and stability.
- **Batch Size**: The number of samples used in each iteration of training.
- **Epochs**: The number of times the entire dataset is used in training.

3.5.6 Other Performance measures

Root Mean Squared Error (RMSE) Root Mean Squared Error (RMSE) can be calculated using the 'mean_squared_error' function in scikit-learn [9][11]

$$\text{RMSE} = \sqrt{\text{mean\_squared\_error}(y_{\text{true}}, y_{\text{pred}})}$$

Mean Absolute Error (MAE) Mean Absolute Error (MAE) is the average of absolute differences between predicted and actual values. It can be used by using sklearn 'mean_absolute_error' function. Unlike MSE, MAE does not square the errors, making it less sensitive to outliers and larger errors. MAE can be calculated as follows [9][11]

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Coefficient of Determination (R^2) The coefficient of determination (R^2) measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It can be used using sklearn function r2_score function: It indicates how well the regression model fits the data compared to a simple mean-based model. R^2 ranges between 0 and 1, with higher values indicating better fit:

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

OR

$$R^2 = 1 - \frac{TSS}{RSS}$$

Where, **TSS** is the Total Sum of Squares, which represents the total variance in the target variable.**RSS** is the Residual Sum of Squares, which represents the unexplained variance after fitting the regression model.[9][11]

4 Methodology/Analysis

4.1 Data Gathering and Prepossessing

4.1.1 Boston

The Boston Housing Dataset is a classic dataset often used for regression house price prediction tasks. It contains various features such as crime rate, nitric oxides concentration, and more, with the target variable being the median value of owner-occupied homes in thousands of dollars. This dataset provides valuable insights into predicting housing prices based on different attributes. [8]

Due to ethical concerns regarding potential bias and discrimination, the original researchers of the Boston Housing Dataset extended it to incorporate ethical considerations, emphasizing diversity and fairness [14]. The full dataset is available at `http://lib.stat.cmu.edu/datasets/boston_corrected.txt`.

Loading the data here i have loaded the dataset in pandas dataframe and named it "data".

Handling duplicates in this case since no duplicates were found we skipped removing them.

```
print(data.duplicated().any())
False
```

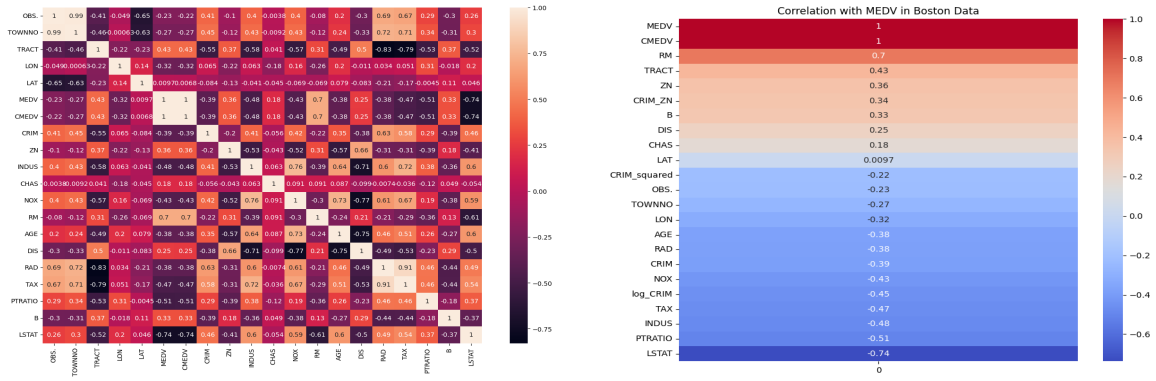
How do we Handling Missing Values? It involves dealing with missing values by using techniques such as imputation, where missing values are replaced by estimated values, or by dropping rows or columns of these missing values. [17] In this data set since there were no missing values i didn't have to perform the above mentioned techniques. The code for it can be found in section E.1

What is Feature Engineering? It is creating new features or transforming existing ones can provide additional information to the model. For instance, transforming skewed distributions to more normal distributions can improve model performance [7]. For this dataset i have we have created 3 new features which are as below, The code for it can be found in section E.2.

1. CRIM_ZN which is a combination of Crime rate and proportion of residential land zoned for lots over 25000 sq.ft.
2. CRIM_squared which is a square of crime rate
3. log_CRIM which is natural log of crime rate

Correlation analysis Here we calculate the correlation matrix between features to identify which variables are positively or negatively correlated with each other. This helps us to understand the inter dependencies between different features. It shows relationship between the features all the features. **What is the correlation features among the boston data? how is target correlated in the boston to all other features?** Here i tried to plot all the numerical columns correlation among the dataset from the figure 3 we can see that that CMED and MED are highly corelated with MEDV while Tract, PTRATIO, INDUS, LSTAT, RM and newly created features above show some correlations with the house price while latitude shows almost no corelation.it's code can be found in section E.5.

Splitting train and test set Here we use the train_test_split() function from sklearn library to split the data into testing and training set in ratio of 75% and 25% respectively. Code can be seen in section E.4.



(a) Correlation of features in Boston

(b) Correlation with target

Figure 3: Correlation plots

Feature Scaling Here i have used MinMaxScaler is a technique used for feature scaling, which transforms the features to a common scale between a specified range, often between 0 and 1. The Boston Housing Dataset contains various features with different scales and magnitudes. Some features may have larger ranges than others, which can lead to certain features dominating the learning process and affecting the model's performance. Also using neural networks using above scalar allows me to converge faster and more reliably when the input features are on a similar scale. Refer code in section E.3.

4.1.2 California

The California Housing Dataset contains data on housing attributes from the 1990 California census. The description of its columns can be found at`https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html`. **Loading the data** here i have loaded the dataset in pandas dataframe and named it "cali".
Handling Duplicates in this case since no duplicates were found i skipped removing them.

```
print(cali.duplicated().any())
False
```

Handling missing values in this case we check for null values within dataset. In this case I'm dropping the rows for missing values as it is a very small number compared to whole dataset. Only total_bedrooms have 207 missing values in them you can see the code for it in F.1.

Feature Engineering Here we are creating new features or transformation from existing features from Boston Housing.
Creating location from Ocean proximity features:Here I have combined back the locations into Near Bay, Near Ocean and Other based on columns <1H OCEAN, INLAND, NEAR OCEAN, NEAR BAY, ISLAND see the code for it in F.2

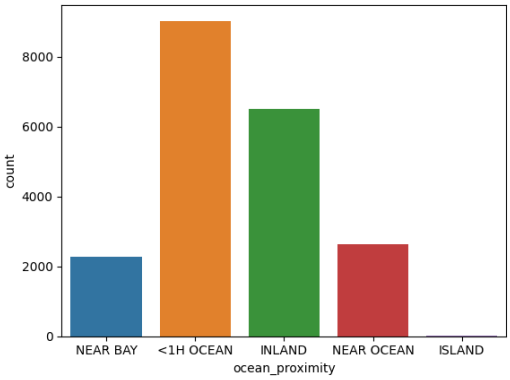


Figure 4: Distribution Plot for for ocean proximity

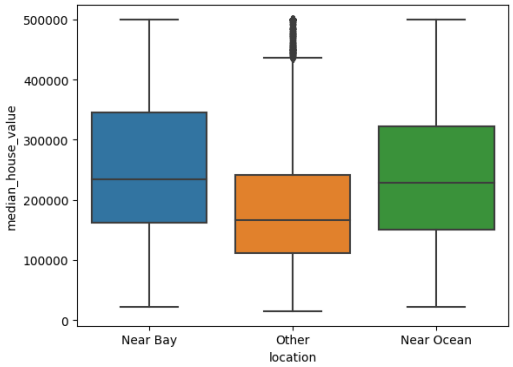


Figure 5: Box plot for location created

Creating income groups: In this case we are dividing the median income into 5 groups as shown in fig 6 you can find the code in F.3.

Creating beds per rooms: Here we apply a transformation to convert the total_bedroom total_rooms ratio. This ratio can be **1** if both are same number of rooms are same and **<1** if house has extra rooms like storage room.

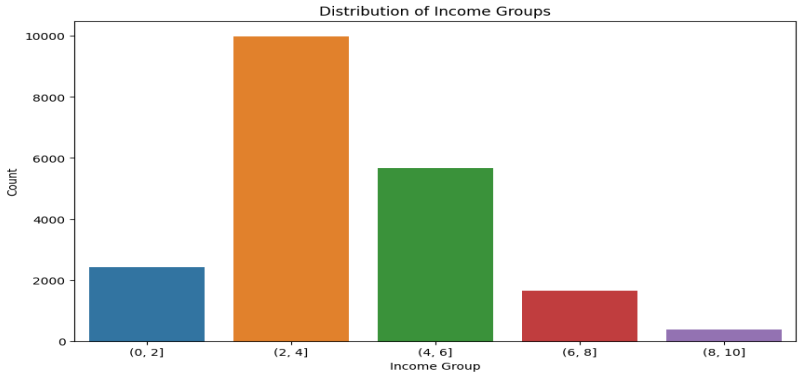


Figure 6: Distribution of median income

```
cali['bedperrooms'] = cali['total_bedrooms']/cali['total_rooms']
```

Correlation Analysis Here we have visualized two corrections first before feature engineering and second after feature engineering. do note we have not shown categorical features such as location in the correlation matrix hence those are not shown. from the figure 33 in section F.4 we can see that ISLAND,total_bedrooms,population,households,longitude have little or zero correlation hence we drop these columns before experiment research.

Feature Scaling Here i have used MinMaxScaler() and defined a custom_scale function because the of dummy variables of <1H OCEAN,NEAR BAY, NEAR OCEAN, INLAND. Refer the code in section F.8

Additionally, I have also done Ames Housing Analysis but due to page limit I have moved it to section .

4.1.3 United Kingdom

This is the Price Paid Data tracks property sales in England and Wales submitted to HM Land Registry for registration. Price Paid Data is based on the raw data released each month. The whole dataset description can be found at `https://www.gov.uk/guidance/about-the-price-paid-data`. I have downloaded the the data copy from kaggle from which contains a single file of all the data from year 1995 to 2017 `https://www.kaggle.com/datasets/hm-land-registry/uk-housing-prices-paid`

Loading the dataset This is a challanging task as the dataset is large if we split the dataset into chuncks and load it one chunk in the memory it causes a lot of cpu and ram computation. To efficiently load the whole data i used pyarrow engine along with pandas as pyarrow engines are faster than python engine and multithreading is currently only supported by the pyarrow engine also it automatically converts date object to datetime object. This is only available with the latest version of pandas as this engine was released in 2022.

Splitting the data for analysis Due to huge data set we are smaller dataframe based on location like london or years like most recent years present in dataset that are 2017 and 2016 for analysis and algorithm implementation

```
london = df [df['Town/City'] == 'LONDON']
dfy17 = df[df['Year']==2017]
dfy16 = df[df['Year']==2016]
```

Dropping single valued and unique columns Here i identified two columns that have completely single valued Record Status - monthly file only and unique Transaction unique identifier. Since all records in Record Status - monthly file only are of A values only we are removing this column as it doesn't make any contribution to Target. Dropping Transaction unique identifier as it's a unique identifier for each transaction which doesn't contribute to target

```
dfy17 = dfy17.drop(columns=['Record Status - monthly file only','Transaction u
dfy16 = dfy16.drop(columns=['Record Status - monthly file only','Transaction u
```

Removing Duplicates Here i have checked for unique values by removing the duplicated rows and only keeping the first distinct rows.

```
dfy17.drop_duplicates( keep='first', inplace=True)
dfy16.drop_duplicates( keep='first', inplace=True)
```

Handling missing values Here i have checked for null values since there are not any null values I'm skipping this step. The code for it is shown in section G.2

Removing outliers Here I'm removing any values which lie outside of two standard deviations σ. The function for removing outliers are shown in section in G.1.

```
Outlier rows removed for 2016 df are 4159
Outlier rows removed for 2017 df are 1554
```

Correlation Analysis This is not really possible for this dataset as most of the columns are categorical see fig 41 in section G.9 we can see that most of the features are either categorical or datetime since we are using numerical values we.

Factorization and splitting into train and test set In this we factorize columns like Town/City,District,County to convert categorical column to numerical column and then we split into train and test set. Code can be seen in section G.4

Feature Scaling Here i have used MinMaxScaler() to scale the features

```
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = pd.DataFrame(scaler.transform(X_train), columns = X_train.colu
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns = X_test.columns
```

4.1.4 Ames

Additionally all the data gathering and preporcessing for Ames are give in section H.1.

4.2 Exploratory Analysis

4.2.1 Boston

Exploratory analysis is a preliminary step in data analysis here we ask hidden questions and some fundamental questions to help explore our understanding the boston dataset. Below are some of the exploratory analysis i have included.

Relational Analysis Here i have visualized how different features relate to each other using scatter plots , boxplots and density plots.

Is there any relationship between different variables and the median value of owner-occupied housing (MEDV), lower status population, crime rate and average rooms? Here i have tried to explore MEDV and other variables such as CRIM (per capita crime rate), RM (average number of rooms per dwelling), and LSTAT (percentage of lower status population). As we can see from figure 7 the first plot on left shows

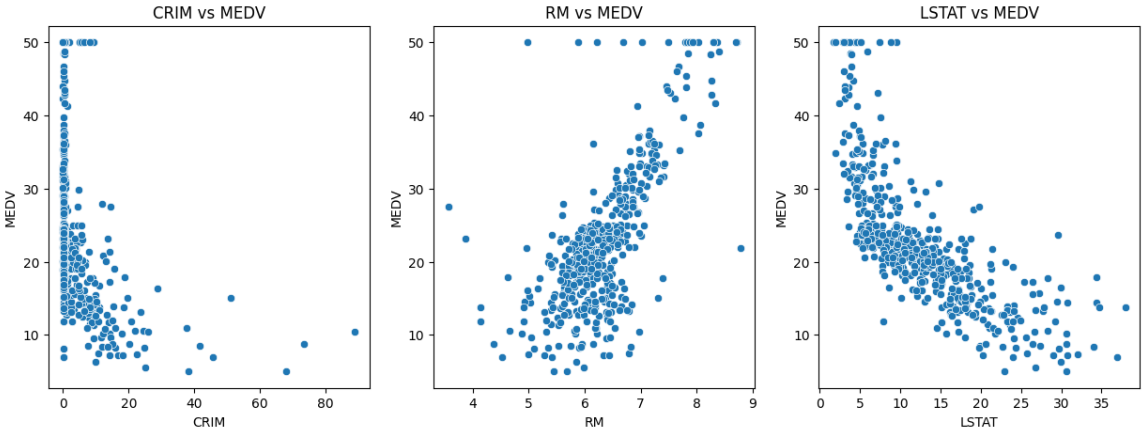


Figure 7: MEDV vs CRIM,RM, LSAT

it appears that there is a negative correlation between the per capita crime rate and the median value of owner-occupied homes. That is as the crime rate increases, the median value of homes tends to decrease. From the second plot shows in the middle it appears that there is a positive correlation between the average number of rooms per dwelling and the median value of owner-occupied homes. Which means as the average number of rooms increases, the median value of homes also tends to increase. From the third plot at the right it appears that there is a negative correlation between the percentage of lower status population and the median value of owner-occupied homes as the percentage of lower status population increases, the median value of homes tends to decrease.

Does the Proximity from river affect house prices? It does even thought Charles River Proximity (CHAS) is not a highly correlated variable when compared to target but when we try to visualize it. We try to visualize how proximity from Charles river have affected the house price from figure 8 we can clearly see that the proximity closer to Charles river have a higher price compared to the once which are further from river.

Outlier Analysis Here we try to plot box plot represents the distribution of values for all numeric feature data-set.Have a in section E.6 figure 29 helps to understand the spread, central tendency, and potential outliers in each features.

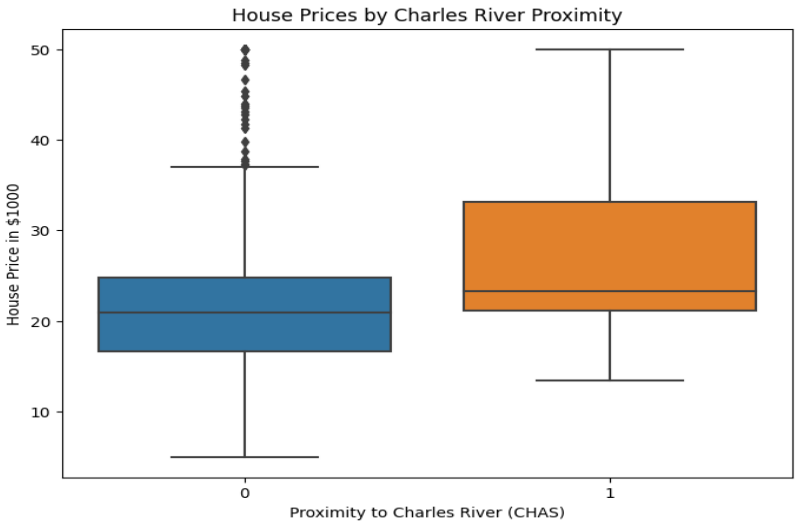


Figure 8: How distance from Charles river affect on house price

4.2.2 California Housing Dataset

Here we perform different explorations to understand the California dataset.

Relational Analysis

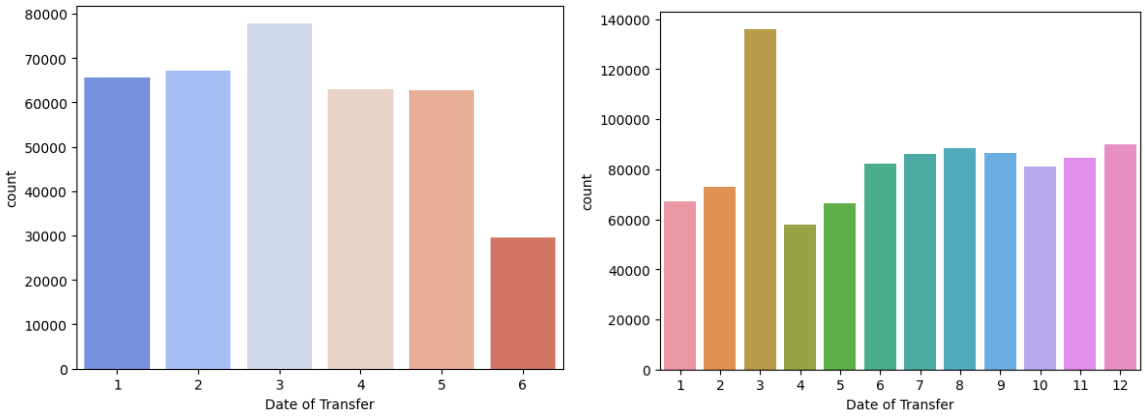
is there a relation in median_income, median_house_price with respect to location? From the visualization 32 in section E.9 we can see that the house prices show an exponential relationship with median income and most of the house prices belong to median_income of $10k-80$k per annum for median house prices ranges from prices \$100k-\$400k and these are ISLAND and <1H Ocean or Inland or Near Ocean. The Near Bay houses are much costlier and prices start from minimum of \$100k.

is there a relation in population with respect to location? The visualization of figure 35 in section F.6 represents relationship between house price and population we can clearly see that lower than 5k population tend to have majority houses while as the population increases house prices tend to decrease house in the most populous area of 20-40k population has a house price around \$118000-\$134000 which are inland or < 1H ocean while those around 10-20k populations prices are around \$28000-\$450000 which are a much broader price range.

What is the kernel density for median_income with respect to median_house_value? Here I have plotted the density of the dataset for median_income with median_house_value. From visualization we can see that most of the data lies in \$20k-\$40k income per annum with house prices ranging from \$100-\$200k

4.2.3 United Kingdom

Which are the months with most transfers? Here we are exploring the Date of transfer for 2017 and 2016. From figure 9 we can see that for 2017 highest number of transfers were done in March then in February followed by January. For 2016 highest number of transfers were done in March then in December followed by August.



(a) 2017 transfers

(b) 2016 transfers

Figure 9: UK Transfers

What are the variations in house price per property type per months Figure 42a in section G.10 shows the difference in prices for each property type. we can observe that detached houses have much higher house prices all year round compared to other property types however other property types grow over a period of march. These other properties include where a property comprises more than one large piece of land which makes sense since harvest grows from spring season so the land of price agricultural lands grows as well.

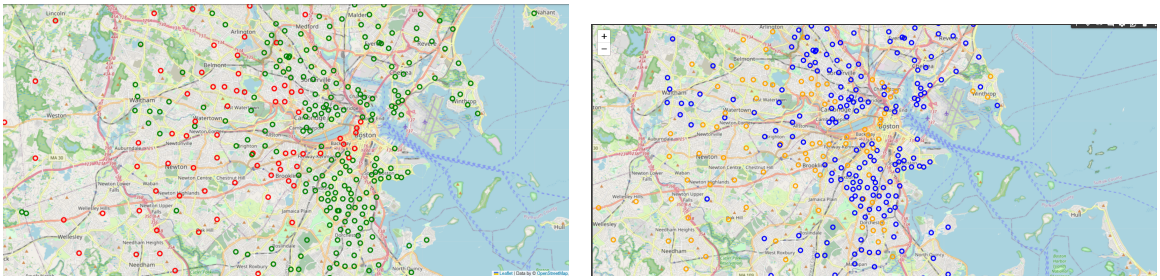
What are the differences in price for individual months? Fig 42b in section G.10 shows per month difference for 2017 and 2016. The price rose mostly during the start of the year since much of months 2017 are not available we cannot surely say that it grew for each month in later months.

Time series Analysis Here we perform time series analysis on London data as shown in fig 36 and code for it can be found in section G.3. Here we have used Auto-regressive Integrated Moving Average (ARIMA) for time series analysis where we are providing house price time series pattern until year 2015 and it makes a forecast for following for 2016 months.

4.2.4 Ames Housing

Additionally I have done Ames housing exploratory analysis which can be seen in section H.1.5.

4.3 Geographic Analysis



(a) High and low priced houses in Boston (b) Average rooms per dwelling added

Figure 10: Geography Plots for Boston

4.3.1 Boston Geographic Analysis

How do we identify high and low priced regions geographically on a map? To decide high price region i took top 75% of the house price which is roughly $25000 and then plotted it on a map to visualize i used circle to mark the regions with green and red anything thats marked as green is lower priced region and red is marked as a high price region. from this we can gather new insights that are house price of property that near airport and city center parks seems to be lower than 25k while outer regions of city like Newton, Watertown, Westwood, Marblehead and Milton have a much higher price than median $25k.

Why are these regions priced so differently? Regions near the airport have a lot of noise and high population density they are older in age and have less amenities due to which they have a lower price band. I did some investigation on these as to why these regions have higher price band and i found out multiple reasons.Firstly these locations have a lot of amenities more schools, parks, and shopping centers and corporate offices. Secondly these regions are at the outskirts of boston the usually have a lower house interest rate which has increased the demand of these regions. Thirdly age of these house is less as they were built later hence these are modern houses which are larger in size and have more amenties like parking, garage and bedrooms due to which they have increased the prices in these regions. while the houses at the city center are usually older and have less facilities than these.

What happens if we change the criteria and include average number of rooms as well? In this case i have added another criteria average number of rooms per dwelling i have found top 75% average rooms that is 6.5 and house price of 25k and marked **blue** if average number of rooms are less than 6.5 and house price is less 25k otherwise marked them **orange** as shown in fig 10b in section E.7.

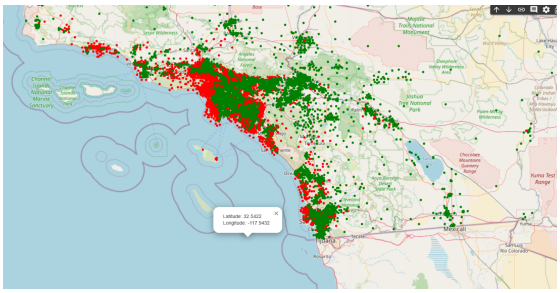
Which are the top 10 Towns with highest, medium, lowered Priced Owners-Occupied houses? Here i have categorizing the median value of owner-occupied homes into three categories: 'low', 'medium', and 'high'. This categorization is based on whether the home's value falls into the bottom 33%, middle 34%, or top 33% of all home values. Second, it's calculating the average home value for each town. Then visualized the counts by sorting the top 10 of each category as we can see in figure 30a in section E.7.

```
data['price_category'] = pd.cut(data['MEDV'], \
bins=[0, np.percentile(data['MEDV'], 33), \
np.percentile(data['MEDV'], 67), np.inf], \
labels=['low', 'medium', 'high'])
town_price=data.groupby('TOWN')['MEDV'].mean().reset_index()
```

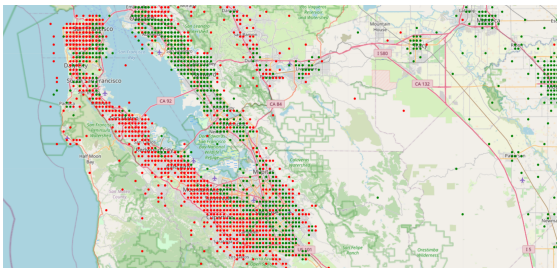
Which are the top 10 cheapest towns to buy the houses from? Here i have analyzed the cheapest towns which have the lowest median house values by sorting and getting lowest 10 towns as shown in fig 30b.

4.3.2 California Geographic Analysis

How do we identify high and low priced regions geographically on a map? Here I have taken top 75% of the house price which is roughly $263900 and marked it's location on the map with red dot if the price is greater than $26k and green dot if it's below $26k. As shown in fig 11a i have identified two regions which were specifically higher priced. These regions of the top region include Redwood city, Mountain View, Cupertino, Saratoga Palo Alto, SunnyVale and San Francisco and the botton region Malibu, Santa Monica, Torrance, Seal beach, Laguna beach and Ocean beachinclude while most of the outskirts of California have cheaper houses. Some of the reasons why these are so high priced is. Firstly, these are at the coast or beaches and are known for their beautfiul locations, and natural attractions. Places like Saratoga, Palo Alto, and Sunnyvale are not only located in desirable landscapes but also provide access to coastal areas. Such features tend to attract affluent buyers willing to pay a premium for unique living environments. Secondly, these regions have a lot of airports near them due to which these are tourist hot spots for vacations as we can see from fig 11b. Thirdly, urban areas often offer a wide range of amenities, cultural attractions, restaurants, and entertainment options. The availability of such amenities enhances the quality of life and makes these regions more desirable to live in Fourthly, Regions like Redwood City, Mountain View, Cupertino, and San Francisco are located closer to major urban centers, business hubs, and employment opportunities Lastly,



(a) Outer view of higher and lower priced regions of California



(b) Zoomed image of the region

Figure 11: Geolocation Plots for California

Regions like Cupertino and Mountain View, are home to major technology companies like Apple and Google

The outskirts of California may have lower housing prices due to factors like greater distance from urban centers, fewer job opportunities, lower demand, and a more suburban lifestyle. These areas might appeal to individuals seeking a country side environment or more affordable options, leading to relatively lower prices.

4.3.3 United Kindom Geographic Analysis

What are the top 15 cities, districts and counties to buy home from in UK? Here we combine and group the data based on citi,district or counties and sort out the top 15 for 2016 and 2017. Here we explore all the different cities, district and counties shown in fig 12

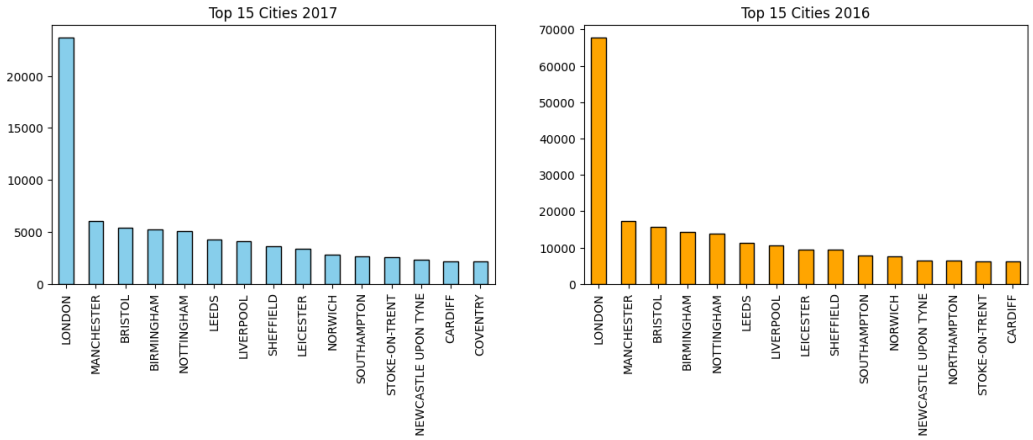


Figure 12: UK top 15 houseprices

How do the median and max investments change in these regions? Figure 13 shows the top 15 cities with total median and max investments.

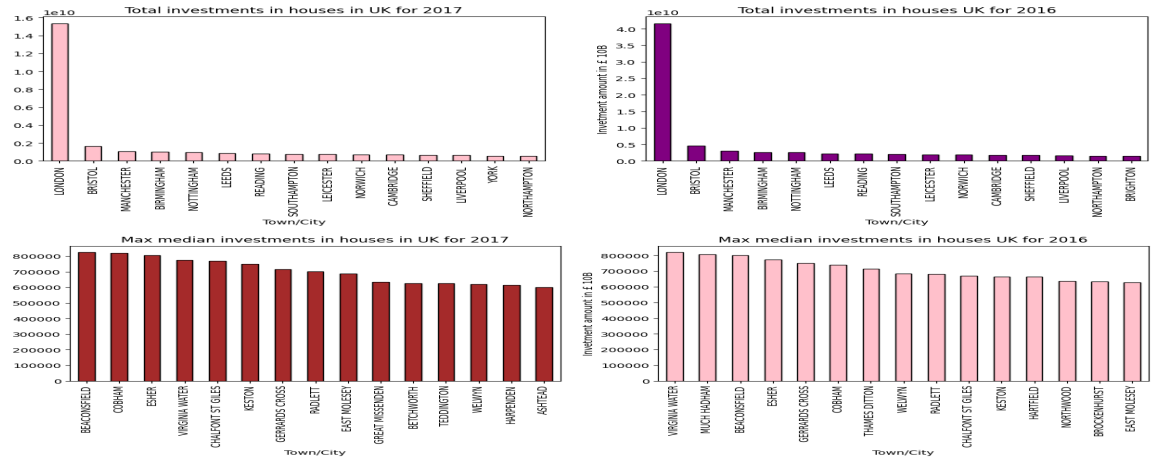


Figure 13: Max and median top 15 investment cities for UK housing

What are the total investments difference in the new and old towns The total investments done for new and old houses in UK is shown in fig 37 in section G.5 we can see that London has the highest investment for both old and new houses which is approximate £1.8B for new houses in 2017 and £1.3B for old houses.

Which are the towns having the max growth? Here we explore the minimum transaction counts growth then the top 10 downs with max growth and new towns growth. The growth is difference in the price of two years. From fig 40 in section G.8 shows growth as we have a criteria if growth is greater than £5 in which Colwyn bay in welsh has the highest investment but smaller towns like Cwmbran and Bingley show the maximum growth and the fig 39 first plot shows top 10 towns showing max growth in which llansanffraid in whales has the highest growth considering overall investments and second plot shows the max growth of new town in which Croydon has the max growth in investment for new towns.It shows some interesting results on how smaller towns show higher growth potentials for property rates compared to larger cities like London.

4.3.4 Ames Geographic Analysis

You can find the Ames housing Geographic analysis in section H.3

4.4 Distribution Analysis

Here we try yo find distribution of values for a numeric feature in the data-set.That means determine if a sample data were drawn from a population that follows a hypothesized

probability distribution. The density plots provide insights into the shape weather these plots follow Gaussian distribution or some other distribution.

4.4.1 Boston

I have separated plots for features and target. If we look at fig 31 in section E.8 shows that the distribution of target variable is not truly Gaussian it seems to have two tails at the right side of the distribution. In the features there are a few columns that follows a Gaussian distribution these are Latitude, Longitude and average number of rooms per dwelling.

4.4.2 California

I have explored the features and target. Refer figure 34 in section F.5 it shows distribution of target variable is not truly Gaussian it seems to have two tails at the right side of the distribution. In the features no columns follows a Gaussian distribution.

4.4.3 United Kingdom

Looking at figure 51 shown in section G.6 shows the distribution of while total area and log sale price seems normally distributed all the other columns are positively or negatively skewed hence we apply scalars to fix the issue. Looking at the target it seems to follow a Gaussian distribution.

4.4.4 Ames

Additionally, I have also done distribution analysis for it which can be found in section H.4.

5 Experimental Results

In this section we will discuss all the modelling and comparisons we did on all the data sets and what we finally decided to proceed for the pipeline.

5.1 Boston Housing Dataset

5.1.1 K-Nearest Neighbors

We apply the K-Nearest Neighbors model to the Boston Housing Dataset with a parameterized grid as follow and got these optimal results.

```
param_grid = {
    'n_neighbors': range(1, 6),
    'distance_type': [1, 2],
    'weighted': [True, False]
}
Optimal hyperparameters: {'n_neighbors': 4,
```

```
'distance_type': 1, 'weighted': True}
Best R2 Score: 0.9124725054570614
```

with these optimal parameters found out that its working best with 4 nearest neighbours, Manhattan distance and weighted distance.

Further experiments With these optimal parameters we experiment with 70 nearest neighbours and see how we the training and testing R^2 vary with these params. From the figure 14 we can see that the model starts to underfit and the training and the test scores both start to decrease after 4 nearest neighbours.

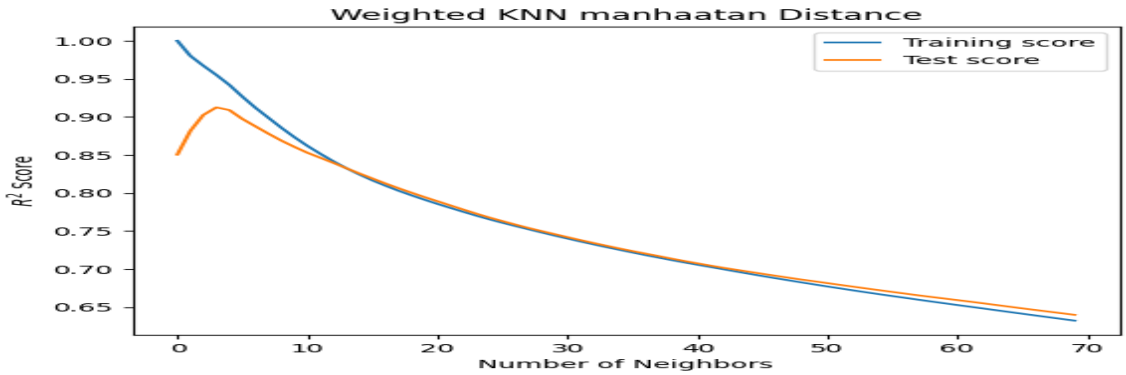


Figure 14: KNN neighbours experiment with optimal hyperparams

5.1.2 Random Forest

Before applying random forest we first need to calculate the max number of features scientifically it was research that it's best to take $\frac{P}{3}$ and ceil up the results for best performance. The max features for boston comes out to be 7 for our dataset.

$$p = int(np.ceil(X\_train.shape[1]/3))$$

Now we apply the RandomForest Regressor and get these optimal hyper parameters

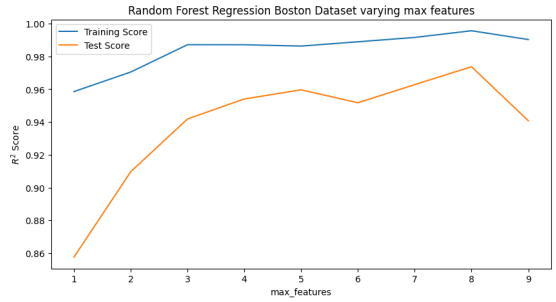
```
param_grid = {
    'min_samples_split': [2, 10, 100],
    'max_depth': [5, 15, 25],
    'n_estimators':[2,5,10,15,25]
}
Best hyperparameters: {'min_samples_split': 2,
'max_depth': 15, 'n_estimators': 15}
Best R2 Score: 0.9799681131527231
```

Further Experiments

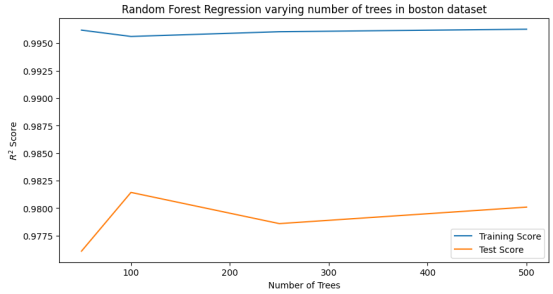
Changing the max features with best hyperparameters Here we vary the max features p from 1-9 features. From the figure 15a graph we can see that as we change the max features there is a constant improvement in the test scores however after 3 features it does not significantly change the results.

Changing the n_estimators with best hyperparameters In this case we experiment with 50, 100, 250,500 n_estimators. From the figure 15b visualization we can see that the model keeps over-fitting after 100 n_estimators.

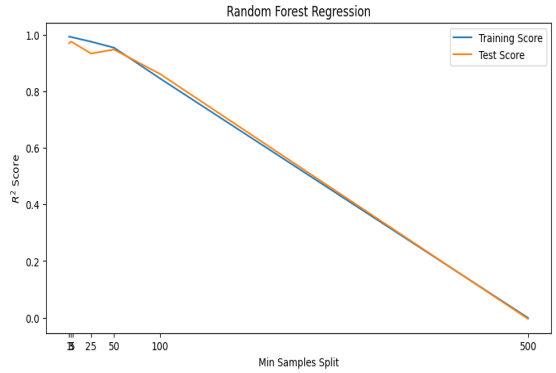
Changing the min_samples_split with best hyperparameters Here we experiment with 1, 3, 5,25, 50, 100,500 of sample splits. From the figure 15c we can see that the model seems to under-fit as it keeps on decreasing testing and training scores.



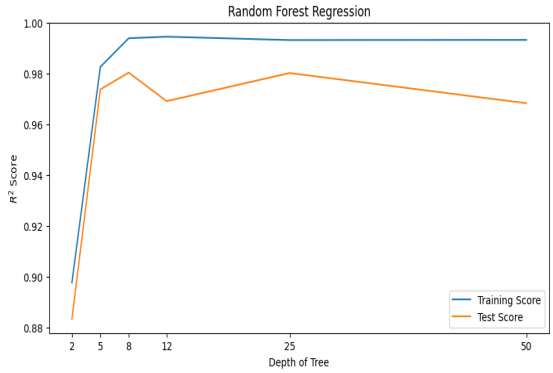
(a) Max Features



(b) N Estimators



(c) Min Samples Split



(d) Max Depth

Figure 15: Hyperparameter Experiments for Boston

Changing the max_depth with best hyperparameters Here we experiment with 2, 5, 8, 12, 25, 50 of max_depths. From the figure 15d we can see that the model but after 8 depth it keeps on overfitting the model.

5.1.3 Multilayered Perceptron

To apply the Multilayered Perceptron model to the Boston Housing Dataset with defaulted parameters to activation sigmoid with two hidden, input layers and 1000 epochs of training

```
n_inputs = X_train_scaled.shape[1]
n_hidden1 = 30
n_hidden2 = 30
n_outputs = 1
epochs = 950 ⟹ loss 2.589792617989705
Test MSE: 2.5576
```

For optimizing the output we apply the MLP Regressor with a parameterized grid and find the optimal hyperparameters shown below.

```
param_grid = { 'n_hidden1': [10, 20, 30],
'n_hidden2': [10, 20, 30],
'learning_rate': [0.001, 0.01, 0.1],
'n_epochs': [100,200,500, 1000] }

Best hyperparameters: {'n_hidden1': 20,
'n_hidden2': 20,
'learning_rate': 0.01, 'n_epochs': 1000}
Best R2 Score: 0.9808
```

Further experiments

Changing the `n_epochs` with Best Hyper-parameters We apply a range of 5000 epochs of training to our MLP model and evaluate its training and testing R^2 score. From the visualization of fig 16, we can observe that both scores remain relatively constant until 500 epochs, after which the model does not learn and scores remain constant. [10]

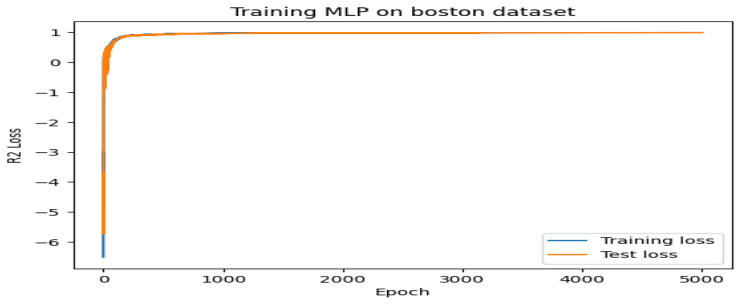


Figure 16: Experimenting epochs with best parameters

5.1.4 Kernelized Ridge Regression

We apply the Kernelized Ridge Regression model to the Boston Housing Dataset with a parameterized grid as follow and got these optimal results.

```
param_grid = {
    'kernel': ['linear', 'polynomial', 'rbf'],
    'alpha': [0.1, 1, 10, 50],
    'degree': [2, 3, 4]
}
Best hyperparameters: {'alpha': 0.1,
'kernel': 'linear',
'degree': 2}
Best $R^2$ Score: 0.9990
```

Now we save the testing scores of each above parameters and visualize from fig 17 the results into 3 separate groups of kernels. The results for each are in E.10 in table 1.

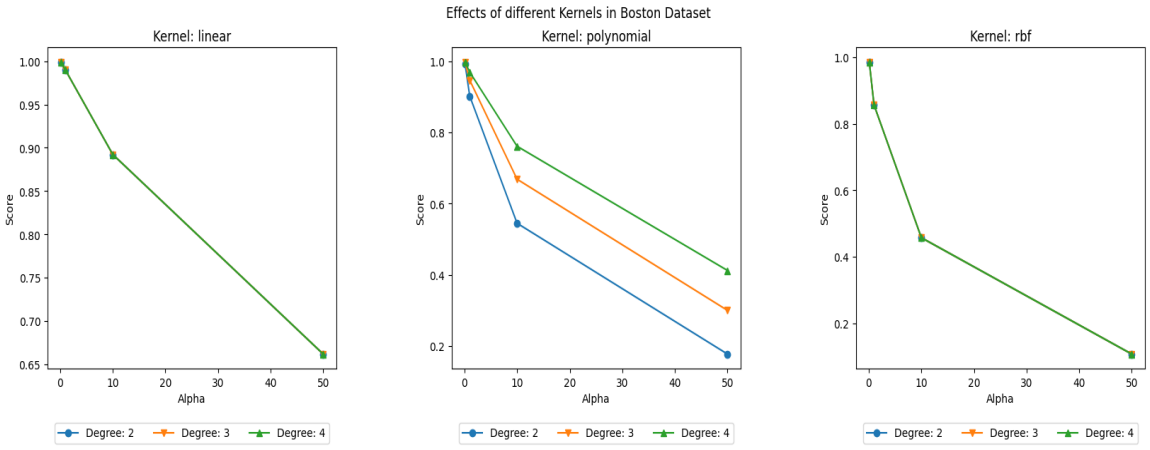


Figure 17: Test R^2 scores for different kernels

5.1.5 Final Comparison

In this part we compare different models performance the best R^2 scores achieved on the Boston housing dataset. Figure 18 shows that Kernelized Ridge Regression outperforms all the other algorithms.

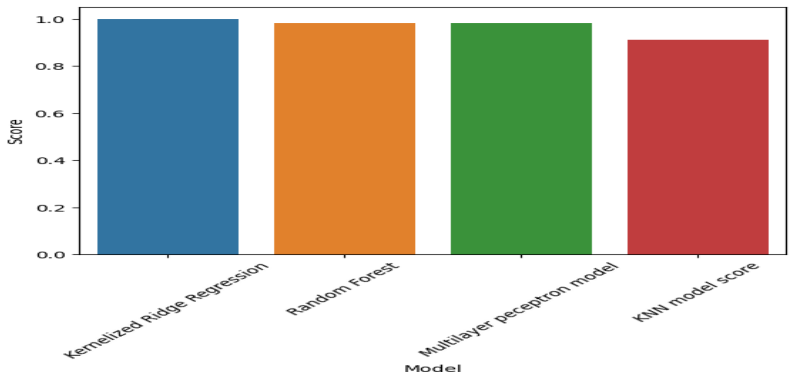


Figure 18: Comparing different models R^2 scores) for Boston

5.2 California Housing Dataset

5.2.1 K-Nearest Neighbors

We apply the K-Nearest Neighbors model to the California Housing Dataset with a parameterized grid as follow and got these optimal results.

```
param_grid = {
    'n_neighbors': range(1, 6),
    'distance_type': [1, 2],
    'weighted': [True, False]
}
Optimal hyperparameters: {'distance_type': 1,
'n_neighbors': 5, 'weighted': False}
Best R2 Score: 0.5508642551703737
```

Further experiments With these optimal parameters we experiment with 7 nearest neighbours and see how we the training and testing R^2 vary with these params. From the fig 19 we can see that the model starts overfits the samples during 1 Nearest Neighbour then starts to evaluate other data-points in consideration and increase its complexity till it reaches to 5 Nearest Neighbours after which it remains constant.

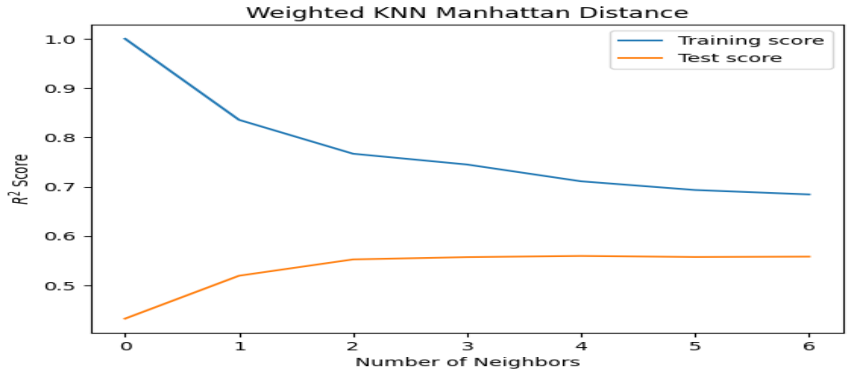


Figure 19: KNN neighbours experiment with optimal hyperparams

with these optimal parameters found out that its working best with 5 nearest neighbours, Manhattan distance and weighted distance

5.2.2 Random Forest

Here firstly we calculate the max number of features. The max features p for california housing dataset comes out to be 3.

$$p = int(np.ceil(X\_train.shape[1]/3))$$

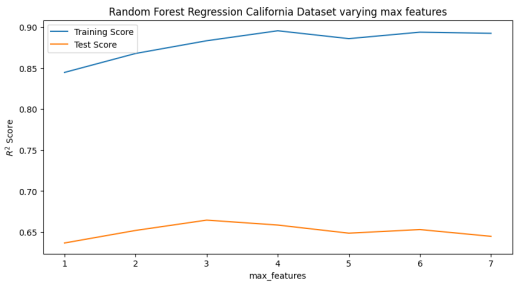
Now we apply the RandomForest Regressor and get these optimal hyperparameters

```
param_grid = {
    'min_samples_split': [2, 10, 100],
    'max_depth': [5, 15, 25],
    'n_estimators':[5,25,50]
}
Best hyperparameters: {'min_samples_split': 10,
'max_depth': 25, 'n_estimators': 25}
Best R2 Score: 0.6233991744484917
```

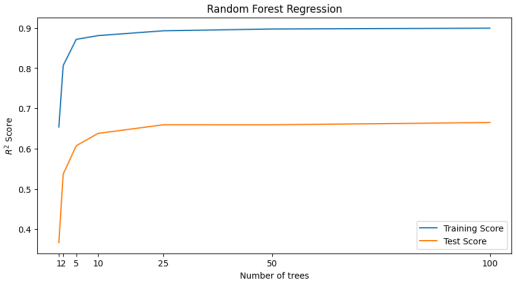
Further Experiments

Changing the max features with best hyperparameters Here we vary the max features p from 1-7 to see it's affect on training and testing scores. From fig 20a we can see that the model is not much affected by changing number of max features as it remains almost constant.

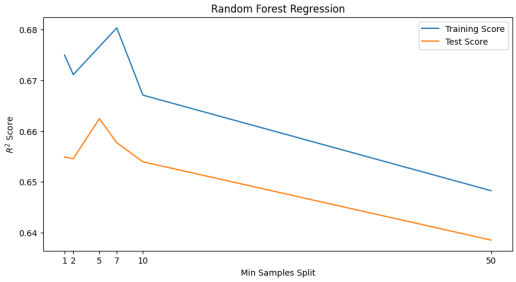
Changing the n_estimators with best hyperparameters In this case we experiment with 1,2,5, 10,25,50,100 n_estimators. From the figure 20b we can see that the model learns from increasing the n_estimators as it significantly improves testing and training scores however after 25 estimators it remains constant and no significant improvement is seen after 25 estimators.



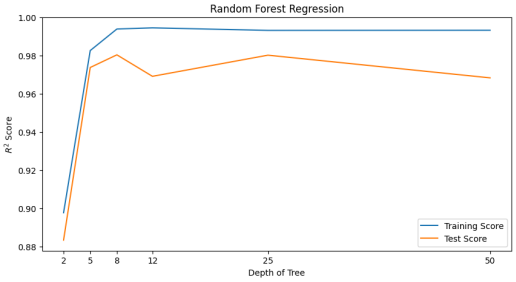
(a) Max Features



(b) N Estimators



(c) Min Samples Split



(d) Max Depth

Figure 20: Hyperparameter Experiments for California

Changing the min_samples_split with best hyper-parameters Here we experiment with 1,2,5,7,10, 50 of sample splits. From the figure 20c we can see that the model learns in the start till 5 min_sample_split it keeps on overfitting the model after that.
Changing the max_depth with best hyperparameters Here we experiment with 2, 5, 8, 25 of max_depths. From figure 20d we can see that the model learns significantly by increasing the depth till it reaches 8 max_depth after that it keeps over-fitting the data.

5.2.3 Multilayered Perceptron

To apply the Multilayered Perceptron model to the California Housing Dataset with activation function relu with two hidden, input layers and 1000 epochs of training

```
n_inputs = X_train_m.shape[1]
n_hidden1 = 30
n_hidden2 = 30
n_outputs = 1
Test MSE: 0.6633
Test R-squared score: 0.5133
```

For optimizing the output we apply the MLP Regressor with a parameterized grid and find the optimal hyperparameters shown below.

```
param_grid = { 'n_hidden1': [10, 15],
```

```
'n_hidden2': [10, 15],
'learning_rate': [0.01, 0.1],
'n_epochs': [ 200,600] }

Best hyperparameters: 'n_hidden1': 15,
'n_hidden2': 10, 'learning_rate': 0.1,
'n_epochs': 600
Best R2 Score: 0.600
```

Further experiments

Changing the `n_epochs` with Best Hyper-parameters We apply a range of 5000 epochs of training to our MLP model and evaluate its training and testing R^2 score. From the figure 21, we can observe that both scores remain relatively close and the model improves underfitting till 1000 epochs after which it does not show significant improvement in scores.

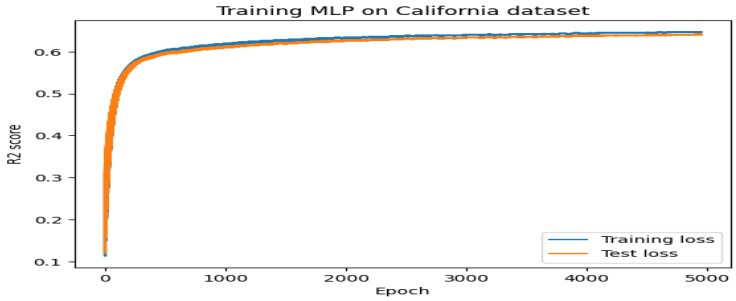


Figure 21: Experimenting epochs with best parameters

5.2.4 Kernelized Ridge Regression

We apply the KernelizedRidgeRegression model to the California housing dataset with a parameterized grid as follow and got these optimal results.

```
param_grid = {
    'kernel': ['linear', 'polynomial', 'rbf'],
    'alpha': [0.01, 0.1, 1,5],
    'degree': [2, 3, 4]
}
Best hyperparameters: {'alpha': 0.01,
'kernel': 'polynomial', 'degree': 4}
Best $R^2$ Score: 0.6639
```

Now we save the testing scores of each above parameters and figure 22 shows the results of three separate groups of kernels. The score results for each are in F.9 in table 2.

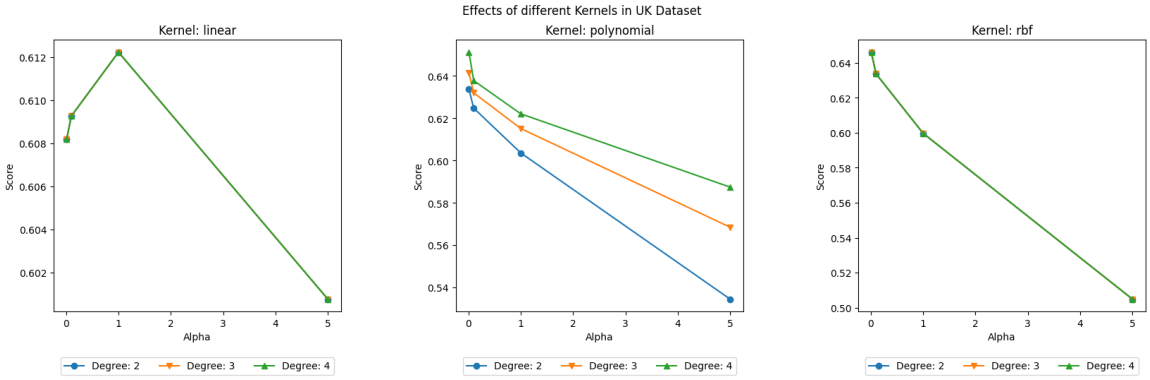


Figure 22: Test R^2 scores for different kernels

5.2.5 Final Comparison

In this part we compare different models performance the best R^2 scores achieved on the California housing dataset. Figure 23 shows that Kernelized Ridge Regression outperforms all the other algorithms for California housing.

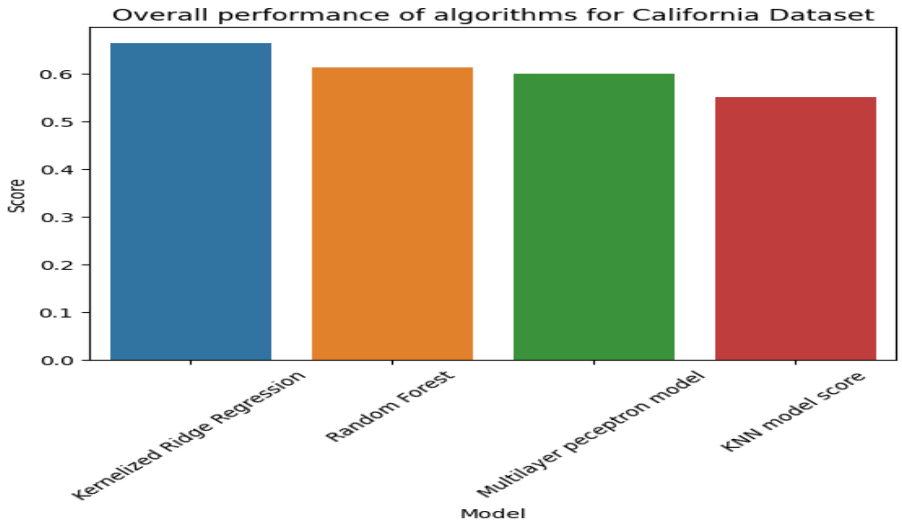


Figure 23: Comparing different models R^2 scores) for California

5.3 United Kingdom

5.3.1 K-Nearest Neighbors

We apply the K-Nearest Neighbors model to the United Kingdom Housing Dataset with a parameterized grid as follow and got these optimal results.

```
param_grid = {
    'n_neighbors': range(1, 6),
    'distance_type': [1, 2],
    'weighted': [True, False]
}
Optimal hyperparameters: {'distance_type': 1,
'n_neighbors': 2, 'weighted': True}
Best R2 Score: 0.5294830929070281
```

Further experiments With these optimal parameters we experiment with 2 nearest neighbours and see how we the training and testing R^2 vary with these params. From the fig 24 we can see that the model starts overfits the samples during 1 Nearest Neighbour then starts to evaluate other data-points in consideration and increase its complexity till it reaches to 2 Nearest Neighbours after that it starts underfitting the data. with these

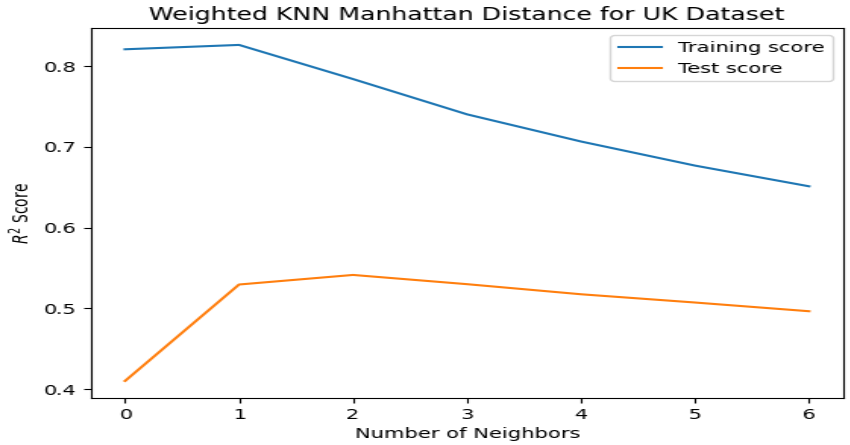


Figure 24: KNN neighbours experiment with optimal hyperparams

optimal parameters found out that its working best with 5 nearest neighbours, Manhattan distance and weighted distance

5.3.2 Random Forest

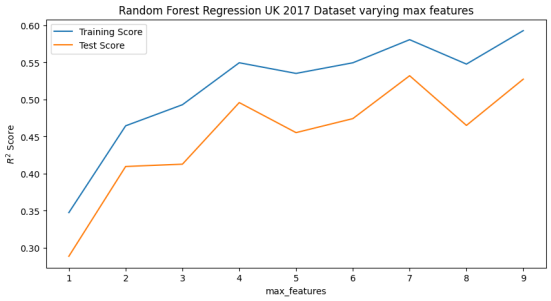
Here firstly we calculate the max number of features. The max features p for United kingdom housing dataset comes out to be 7.

$$p = int(np.ceil(X\_train.shape[1]/3))$$

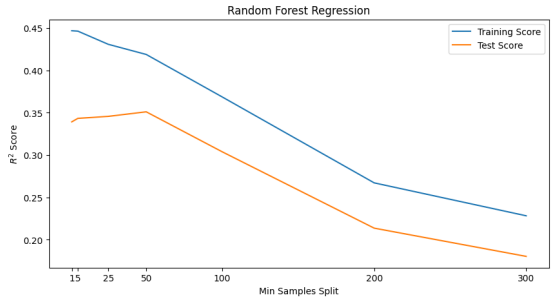
Now we apply the RandomForest Regressor and get these optimal hyperparameters

```
param_grid = {
```

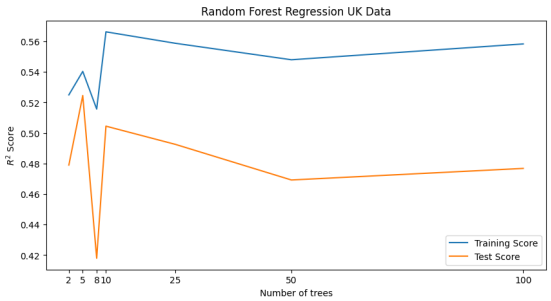
```
    'min_samples_split': [2, 10, 100],
    'max_depth': [5, 15, 25],
    'n_estimators':[5,25,50]
}
Best hyperparameters: {'min_samples_split': 50,
'max_depth': 10, 'n_estimators': 10}
Best R2 Score: 0.5052426703746988
```



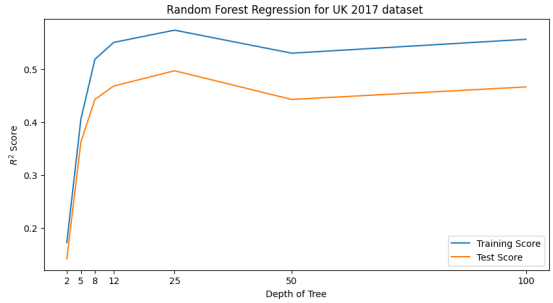
(a) Max Features



(b) Min Samples Split



(c) N Estimators



(d) Max Depth

Figure 25: Hyperparameter Experiments for UK

Further Experiments

Changing the max features with best hyperparameters Here we vary the max features p from 1-9 to see it's affect on training and testing scores. From fig 25a we can see that the model is over-fitting till 7 features after which it keeps under-fitting the data.

Changing the n_estimators with best hyperparameters In this case we experiment with 1,2,5, 10,25,50,100 n_estimators. From the figure 25c we can see that the model learns till 5 n_estimators as it significantly improves testing and training scores after 5 estimators it keeps under-fitting the data till 50 and again over fitting till 100 estimators.

Changing the min_samples_split with best hyper-parameters Here we experiment with 5,25, 50, 100, 200,300 of sample splits. From the figure 25b we can see that the model learns in the start till 5 min_sample_split it keeps on over fitting the model after that.

Changing the max_depth with best hyperparameters Here we experiment with 2,

5, 8, 12,25,50,100 of max_depths. From figure 25d we can see that the model learns significantly by increasing the depth till it reaches 25 max_depth after that it keeps over-fitting the data till 50 and it slightly increases till 100 depth then remains constant.

5.3.3 Multilayered Perceptron

To apply the Multilayered Perceptron model to the United Kingdom Housing Dataset with activation function relu with two hidden, input layers and 1000 epochs of training

```
n_inputs = X_train.shape[1]
n_hidden1 = 30
n_hidden2 = 30
n_outputs = 1
Test MSE: 104445606596.4982
Test R-squared score: -0.0509
```

It seems that due to the nature of data we are unable to fit the MLP model. Since we are unable to fit the model we are dismissing the MLP model for final comparisons.

5.3.4 Kernelized Ridge Regression

On applying the Kernelized RidgeRegression model to the United Kingdom housing dataset with a parameterized grid as follows I got these optimal results.

```
param_grid = {
    'kernel': ['linear', 'polynomial', 'rbf'],
    'alpha': [0.1, 1, 10, 50],
    'degree': [2, 3, 4]
}
Best hyperparameters: {'alpha': 0.1,
'kernel': 'polynomial', 'degree': 4}
Best $R^2$ Score: 0.1678
```

Now we save the testing scores of each above parameters and figure 26 shows the results of three separate groups of kernels. The score results for each are in G.11 in table 3.

5.3.5 Final Comparison

In this part we compare different models performance the best R^2 scores achieved on the United Kingdom housing dataset. Figure 27 shows that K-Nearest neighbour outperforms all the other algorithms for uk housing.

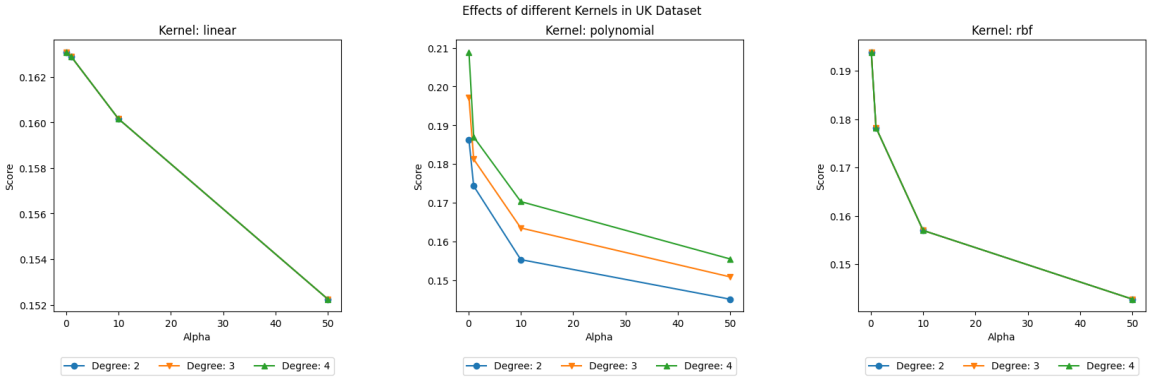


Figure 26: Test R^2 scores for different kernels

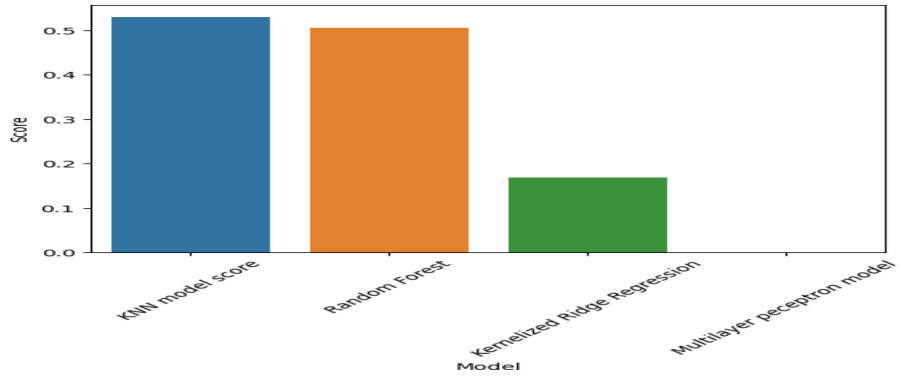


Figure 27: Comparing different models R^2 scores) for United Kingdom

5.4 Ames Housing Dataset

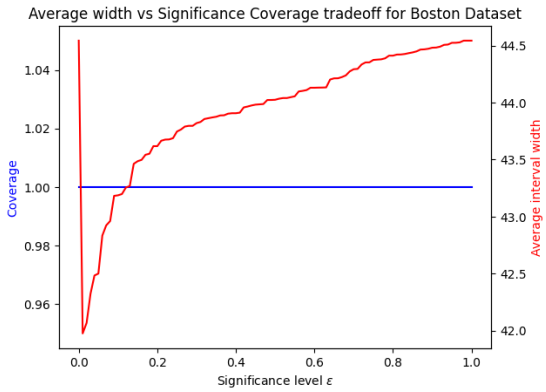
Additionally I have also done experimental result for these 4 algorithms you can find it in section H.5.1.

6 Pipe-lining and Conformal Validity

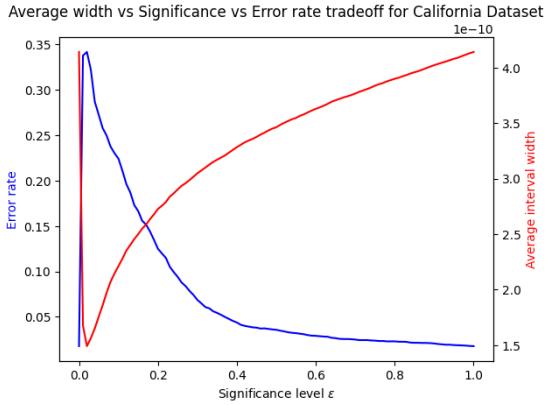
For building out pipeline we apply all the transformations in the previous steps. In this case we would be using sklearn.pipeline Pipeline function to build the pipeline and then we get the y_pred from the pipeline to test it's performance we use r2_score from sklearn sklearn.metrics.

To get the inductive conformal validity of the we apply inductive conformal predictor on the pipeline. Before applying out model we separate our training set into into calibration and training set proper. Then we apply our inductive conformal predictor for a significance level in from 0 to 100%. Then we find the upper and lower bounds, interval, coverage, width

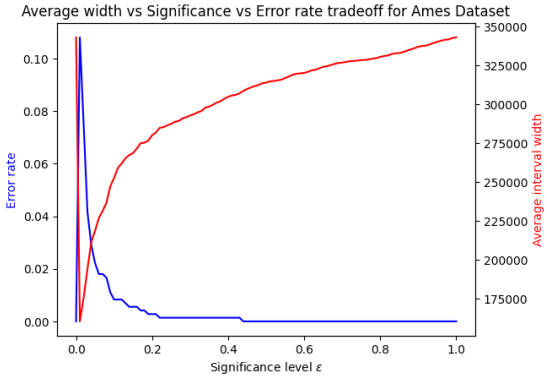
and average width I.



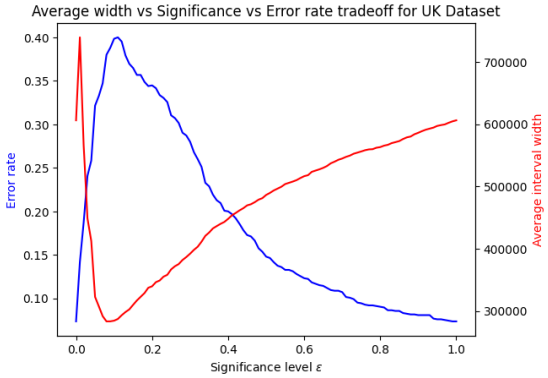
(a) Inductive conformal validity for the pipeline for Boston



(b) Inductive conformal validity for the pipeline for California



(c) Inductive conformal validity for the pipeline for Ames



(d) Inductive conformal validity for the pipeline for UK

Figure 28: Inductive Conformal Validity Plots

6.1 Boston Housing Pipeline

We apply our best models with best hyper-parameters of Boston housing dataset and build our Kernelized Ridge regression pipeline. The code for the pipeline can be seen in section E.11.

R2 score **for** the pipeline: 0.9983156668963727

6.1.1 Inductive Conformal validity for Boston

From figure 28a we can see that the coverage does not change for Boston housing dataset while average with keeps on increasing significance level ϵwhich provides a validity for the

above pipeline to be under inductive conformal.

6.2 California Hosuing Pipeline

We apply our best models with best hyper-parameters of Califonia housing dataset and build our Kernelized Ridge regression pipeline . The code for the pipeline can be found in section F.10.

```
R2 score for the pipeline: 0.6763441141845559
```

6.2.1 Inductive Conformal validity for California

From visualization of figure 28b we can see that the error rate and average with are inversely proportional and at 5% significance we have maximum error rate and lowest average width on increasing significance level ϵ we see error rate increase and width decrease which provides a validity for the above pipeline to be under inductive conformal.

6.3 United Kingdom Housing Pipeline

We apply our best models with best hyper-parameters of United Kingdom housing dataset and build our Kernelized Ridge regression pipeline. Refer the code in section G.12

```
R2 score for the pipeline: 0.5132847607354881
```

6.3.1 Inductive Conformal validity for United Kingdom

From visualization of figure 28d we can see that the error rate and average with are inversely proportional and at 10% significance we have maximum error rate and lowest average width on increasing significance level ϵ we see error rate decrease and width increase which provides a validity for the above pipeline to be under inductive conformal.

6.4 Ames Housing Pipeline and Inductive Conformal validity

Additionally I have done Ames housing Pipeline and inductive conformal validity which can be seen here H.5.6

7 Conclusions and future experiments

The thesis adds to a simple approach of explaining of research of machine learning application in the housing price market. It is important to appropriately evaluate the value of a house. When banks and other financial institutions make loan decisions. These organizations must ensure that the property values they are appraising are proportional in order to prevent losing money on defaulted loans to the loan amount. Lenders usually request an

inspection and appraisal during the loan application process. During the extensive examination of house valuation, several aspects were considered. For future work on this project, we could explore the following possibilities.

7.1 Finding more housing datasets

Here we would like to explore more recent datasets, for example UK Home Land house prices released every year `https://www.gov.uk/government/news/uk-house-price-index-for-august-2021`. Federal housing finance agency houses prices released at `https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index-Datasets.aspx`.

7.2 Implementing more ML models

Here we could explore the use of other machine learning models such as support vector machine with kernels, deep learning network and more advanced regression models like auto regressive models or GPU with RAPIDS which improve the predictions many folds.

7.3 Implement Ensemble

we can include ensemble methods like Bagging and Boosting algorithms like AdaBoost or XGBoost to increase the models performances and sensitivity to outliers.

7.4 Findings by domain experts

Finding and discussing the problems of house prices with mortgage experts or realtors and understanding the parameters on which the prices get affected based on trend or market needs.

8 Self Assessment

Reflecting on the journey of this project, I can confidently say that I have grown both as a researcher and as an individual. Initially, I faced challenges that demanded me to delve into unfamiliar territories. One notable hurdle was my limited familiarity with LaTeX, a vital tool for documenting research. However, this challenge turned into an opportunity for me to expand my skill set. I dedicated time to learn LaTeXand, as a result, enhanced my technical writing skills considerably. This not only enabled me to produce a well-structured and visually appealing report but also prepared me for future research endeavors where documentation is a crucial aspect.

Throughout the project, I discovered that the more about machine learning and data analysis and its complexities. The technical nuances and intricacies of various algorithms often required thorough research and hands-on experimentation. Dealing with errors and unexpected results proved to be a valuable learning experience, teaching me patience and persistence in problem-solving. To navigate through the project more effectively, I adopted

an organized approach, outlining my objectives and planning each step in advance. Regular meetings with my supervisor helped me stay on track, receive guidance, and incorporate feedback to refine my work. This iterative process of refining and optimizing contributed significantly to the quality of my results.

Furthermore, I've learned the importance of addressing ethical considerations in datasets. Recognizing potential biases and discrimination issues within the Boston dataset, I appreciated the significance of incorporating ethical perspectives into my analysis, promoting diversity, and ensuring fairness. I acknowledge that maintaining motivation was crucial, especially during moments when the project presented challenges. Setting milestones helped me stay motivated and focused on the larger picture.

In conclusion, this project not only deepened my understanding of machine learning and data analysis but also instilled valuable skills such as technical writing, problem-solving, organization, and ethical awareness. As I move forward, I intend to apply the lessons learned from this project to future endeavors, leveraging my newfound skills and experiences to contribute effectively to the field of data science

9 Professional Issues

In this section, I will be stating some professional issues that I faced while developing the project. The research journey for this project has revealed a series of challenges and considerations that have greatly impacted its development and outcomes. One of the initial ethical problems encountered was related to potential bias and discrimination within the Boston Housing Dataset. While the dataset was extended to incorporate ethical considerations, such as diversity and fairness, it remains imperfect, raising concerns about the potential for unintended biases to influence the results [14].

Another significant hurdle was the issue of long running times for certain portions of the code. The utilization of algorithms like K-Nearest Neighbors and decision trees led to extensive runtime requirements, sometimes spanning up to 6 hours or more. To mitigate this, I opted to work with smaller data batches, containing around 5000 samples each from California, Ames, and the UK, allowing me to adapt the algorithms more effectively.However, even with smaller data batches, the problem of lacking computational power persisted. Algorithms like Kernelized Ridge Regression and Multilayered Perceptron proved to be computationally intensive and frequently overwhelmed my local system, causing crashes and out-of-memory errors. This forced me to shift my code to the cloud-based Google Colab environment. The UK housing dataset, for instance, demanded up to **10GB** of RAM during dataset loading, underscoring the need for robust computing resources. To comfortably run these codes, I recommend using a local system equipped with at least **16GB** DDR4 RAM or utilizing platforms like Google Colab.

Furthermore, the challenges extended to algorithm compatibility with specific datasets. For instance, the Multilayered Perceptron (MLP) model struggled to fit the Ames and UK dataset effectively, resulting in dismaying zero or negative R^2 scores. In addition to these technical hurdles, I encountered a lack of prior experience with report writing in LaTeX.

This research endeavor became an opportunity for me to learn the intricacies of report composition using LaTeX, and I strove to incorporate my learning to the best of my ability. A notable constraint throughout the project was the scarcity of comprehensive and relevant resources. Specifically, data that could shed light on factors like national bank mortgage rates, popularity of certain areas, impact of amenities like schools and supermarkets, and seasonal variations in house prices was largely absent. While I managed to incorporate year and time series analysis into the UK and Ames housing price assessments, such resources were lacking for datasets like Boston and California.

References

[1] Adyan Alfiyatin, Ruth Ema, Hilman Taufiq, and Wayan Mahmudy. Modeling house price prediction using regression analysis and particle swarm optimization case study : Malang, east java, indonesia. *International Journal of Advanced Computer Science and Applications*, 8, 01 2017.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] J. Brownlee. Difference between classification and regression in machine learning. `https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/`, 2017.

[6] Royal Holloway University of London Centre for Reliable Machine Learning, Department of Computer Science. Conformal Prediction in Classification. `https://cml.rhul.ac.uk/cp.html#/`.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[8] D. Harrison and D.L Rubinfeld. Boston housing dataset. `https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, second edition, 2009.

[10] Andreas C Müller and Sarah Guido. *Introduction to Machine Learning with Python*. O'Reilly Media, Inc., 2016.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,

M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Veysel Murat İstemihan GENÇ Peyman BEYRANVAND, Zehra ÇATALTEPE. Multilabel mlp architecture. `https://www.researchgate.net/figure/Multilabel-MLP-architecture_fig1_327984744`.

[13] HM Land Registry. Uk housing prices paid. `https://www.kaggle.com/hm-land-registry/uk-housing-prices-paid`.

[14] Ayush Varma, Abhijit Sarma, Sagar Doshi, and Rohini Nair. House price prediction using machine learning and neural networks. April 2018.

[15] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[16] Kelvyn Jones Yingyu Feng. Comparing multilevel modelling and artificial neural networks in house price prediction. `https://research-information.bris.ac.uk/en/publications/comparing-multilevel-modelling-and-artificial-neural-networks-in-`.

[17] A. Zollanvari. *Machine Learning with Python: Theory and Implementation*. Springer International Publishing, 2023.

Appendix

A Applied Predictive Modeling

Applying predictive modeling is a technique used to make predictions about future events or behaviors based on historical data. In the context of house price prediction, applied predictive modeling can be used to analyze various factors that affect house prices, such as location, size, and amenities, and use this information to make accurate predictions about future house prices. There are several advanced modeling approaches that can be employed to model house prices, including Artificial Neural Networks. These approaches, along with the standard Hedonic Price Model, can be compared in terms of predictive accuracy, capability to capture location information, and their explanatory power. [16] For example, one study used multiple linear regression to analyze the major factors affecting housing prices and selected significant factors influencing general housing prices. The study then established a multiple linear regression model for housing price prediction and applied the data set of real estate prices in Boston to test the method. The results showed that the multiple linear regression model could effectively predict and analyze housing prices to some extent.

By analyzing data on various factors that influence the price of a house, such as physical conditions, concept, and location, predictive models can help developers determine the selling price of a house and can help customers to arrange the right time to purchase a

house [1]. Our approach in this study to creating a house price prediction model is to use regression analysis and predictive modeling techniques.

B Other regression losses

Huber Loss It is a combination of MSE and MAE. It uses squared errors for smaller errors and absolute errors for larger errors, making it less sensitive to outliers than MSE but more sensitive than MAE. Its formula is given by

$$\text{Huber Loss} = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}\frac{1}{2}(y_i - f(x_i))^2, & \text{if } |y_i - f(x_i)| \leq \delta \\ \delta|y_i - f(x_i)| - \frac{1}{2}\delta^2, & \text{otherwise}\end{cases}$$

Log-Cosh Loss The Log Cosh loss is calculated by the average of the logarithm of the hyperbolic cosine of the differences between predicted and true values. It is less sensitive to outliers than MSE and MAE. Its formula is given by

$$\text{Log-Cosh Loss} = \frac{1}{n}\sum_{i=1}^{n}\log(\cosh(y_i - f(x_i)))$$

Quantile Loss This loss function is used for quantile regression, the goal is to predict a specific quantile of the target variable. It calculates the weighted sum of absolute differences between predicted and true values, where the weights depend on whether the prediction is above or below the true value. Its formula is given by

$$\text{Quantile Loss} = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}\tau \cdot (y_i - f(x_i)), & \text{if } y_i - f(x_i) > 0 \\ (\tau - 1) \cdot (y_i - f(x_i)), & \text{otherwise}\end{cases}$$

C Additional activation function

C.1 Leaky ReLU

The Leaky ReLU activation function is a variant of ReLU that allows a small gradient for negative inputs:

$$\text{Leaky ReLU}(x) = \begin{cases}x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise}\end{cases}$$

where α is a small positive constant. Leaky ReLU addresses the dying ReLU problem that can occur when gradients become zero for all negative inputs.

C.2 Softmax Activation Function

The softmax activation function is used in the output layer for multi-class classification problems. It converts raw scores into a probability distribution across classes. Given $z_1, z_2, \ldots, z_K$ as the raw scores, the softmax function is defined as:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

D Random Forest Structure and Hyper parameters

```
Random Forest
|--- Decision Tree 1
|    |--- Node A
|    |    |--- Leaf 1
|    |    |--- Leaf 2
|    |--- Node B
|         |--- Leaf 3
|         |--- Leaf 4
|
|--- Decision Tree 2
|    |--- Node C
|    |    |--- Leaf 5
|    |    |--- Leaf 6
|    |--- Node D
|         |--- Leaf 7
|         |--- Leaf 8
|
|--- Decision Tree 3
     |--- Node E
     |    |--- Leaf 9
     |    |--- Leaf 10
     |--- Node F
          |--- Leaf 11
          |--- Leaf 12
```

D.1 Hyper parameters in Random forests

Number of Estimators (`n_estimators`) The hyperparameter `n_estimators` represents the number of decision trees in the forest. Increasing the number of estimators can enhance the model's performance, but it may also lead to longer training times and increased complexity [4].

Minimum Samples for Split (`min_samples_split`) The hyperparameter `min_samples_split` defines the minimum number of samples required to split an internal node during tree construction. Setting a higher value can prevent the tree from becoming too deep and overfitting the training data. However, excessively high values might result in underfitting [4].

Minimum Samples for Leaf (`min_samples_leaf`) The hyperparameter `min_samples_leaf` specifies the minimum number of samples required to be in a leaf node. This parameter prevents small leaves that might capture noise. Similar to `min_samples_split`, setting an appropriate value for `min_samples_leaf` helps in controlling overfitting [4].

Other hyper-parameters

- `max_features`: The number of features to consider when looking for the best split.
- `max_depth`: The maximum depth of each tree in the forest.
- `bootstrap`: Whether to use bootstrap samples when building trees.
- `criterion`: The function to measure the quality of a split [4].

D.2

Some other Kernel Ridge Hyper parameters The additional hyper parameters specific to Kernelized Ridge Regression are α scaling factor for the kernel matrix, β Bias term added to the kernel matrix.ϵ small constant to ensure numerical stability in the matrix operations.

E Boston Housing

E.1 Handling missing values code

```
data.isnull().sum()
```

OBS.	0
TOWN	0
TOWNNO	0
TRACT	0
LON	0
LAT	0
MEDV	0
CMEDV	0
CRIM	0
ZN	0
INDUS	0
CHAS	0

```
NOX         0
RM          0
AGE         0
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT       0
dtype: int64
```

E.2 Feature Engineering code

```
data['CRIM_ZN'] = data['CRIM'] * data['ZN']
data['CRIM_squared'] = data['CRIM'] ** 2
data['log_CRIM'] = np.log(data['CRIM'])
```

E.3 Feature scaling code

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(\
X_train.select_dtypes(include=['int64', 'float64']))
X_test_scaled = scaler.transform(\
X_test.select_dtypes(include=['int64', 'float64']))
y_train.reset_index(drop = True, inplace=True)
y_test.reset_index(drop = True, inplace=True)
```

E.4 Train test split code

```
X_train, X_test, y_train, y_test =
train_test_split(data.drop('MEDV', axis = 1)
, data['MEDV'], random_state = 1501)
```

E.5 Correction analysis code

```
fig, ax = plt.subplots(figsize = (15,10))
ax= sns.heatmap(data.select_dtypes\
(include=['int64', 'float64']).corr(),annot = True,);
```

E.6 Outlier Analysis Image

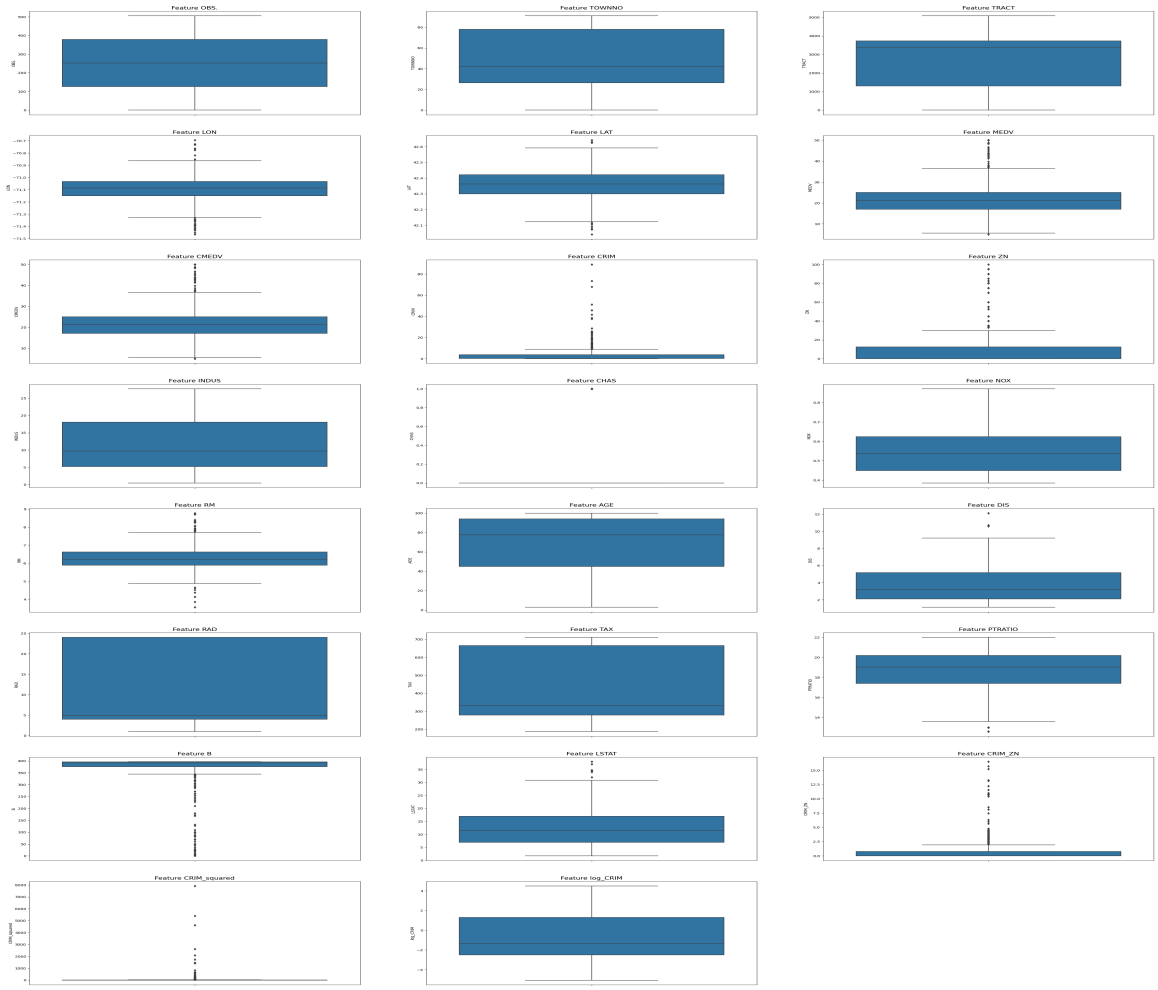
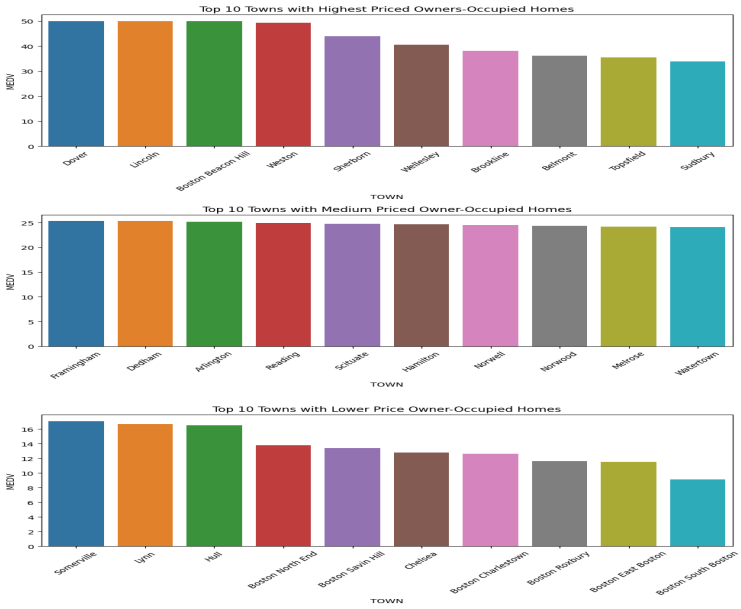
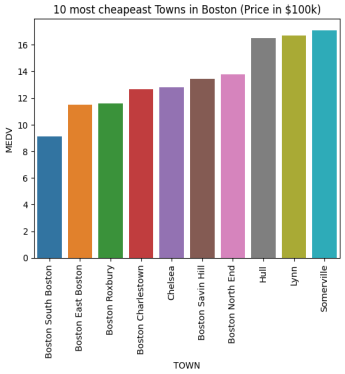


Figure 29: Outlier Analysis of all features from Boston

E.7 Geolocation Analysis for top 10 houses image



(a) Top 10 highest, medium and lowest valued towns



(b) Top 10 cheapest valued towns

Figure 30: Geography Plots for Boston

E.8 Distibution Analysis Image

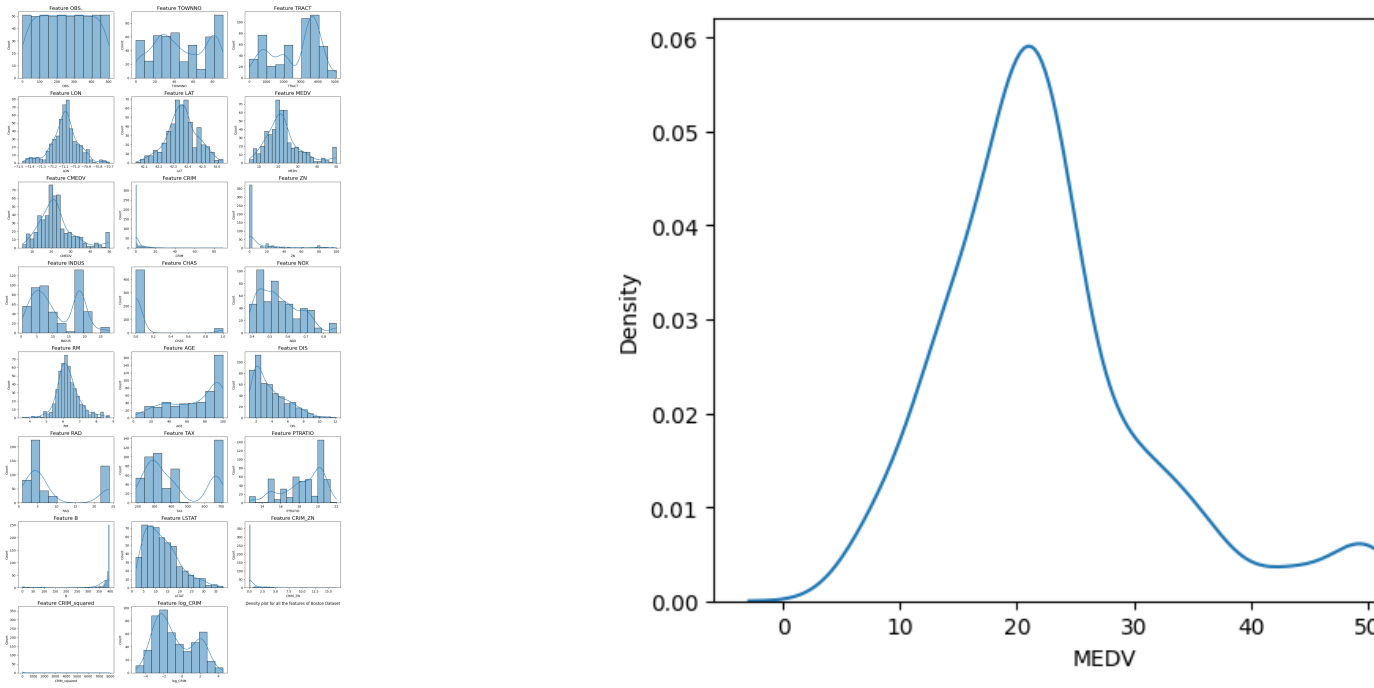


Figure 31: Features and target distribution for Boston dataset

E.9 Relation of median_income,median_house_price with respect to location image

Figure 32: Relationship between median_income and median_house_value with respect to location

E.10 Kernelized Ridge regression table

Kernel	Degree	$\alpha = 0.1$	$\alpha = 1.0$	$\alpha = 10.0$	$\alpha = 50.0$
linear	2	0.999019	0.989704	0.892263	0.661814
linear	3	0.999019	0.989704	0.892263	0.661814
linear	4	0.999019	0.989704	0.892263	0.661814
polynomial	2	0.992772	0.902360	0.544944	0.177166
polynomial	3	0.997109	0.946973	0.668763	0.299986
polynomial	4	0.998143	0.969725	0.761379	0.412178
rbf	2	0.985396	0.856312	0.457027	0.107126
rbf	3	0.985396	0.856312	0.457027	0.107126
rbf	4	0.985396	0.856312	0.457027	0.107126

Table 1: Kernalized ridge for boston dataset

E.11 Pipeline code

```
pipe = Pipeline([
    ('scaler', MinMaxScaler()),
    ('regressor', KernelizedRidgeRegression(alpha=0.1,
                  kernel='linear', degree=2))])
```

F California Housing

F.1 Handling missing values code

```
cali.isna().sum()
longitude                 0
latitude                  0
housing_median_age        0
total_rooms               0
total_bedrooms          207
population                0
households                0
median_income             0
median_house_value        0
ocean_proximity           0
dtype: int64
cali.dropna(inplace=True)
```

F.2 Creating location from Ocean proximity features code

```
cali['ocean_proximity'].value_counts()
<1H OCEAN      9034
INLAND         6496
NEAR OCEAN     2628
NEAR BAY       2270
ISLAND            5
cali=cali.join(pd.get_dummies(cali['ocean_proximity']
)).drop('ocean_proximity',axis=1)
```

F.3 Creating income groups code

```
cali['income_group'] = pd.cut(cali['median_income'],
bins=[0, 2, 4, 6, 8, 10])
grouped = cali.groupby('income_group')['median_house_value'].mean().reset_index
grouped['income_group'] = grouped['income_group']\
.apply(lambda x: x.mid)
```

F.4 Correlation Analysis image

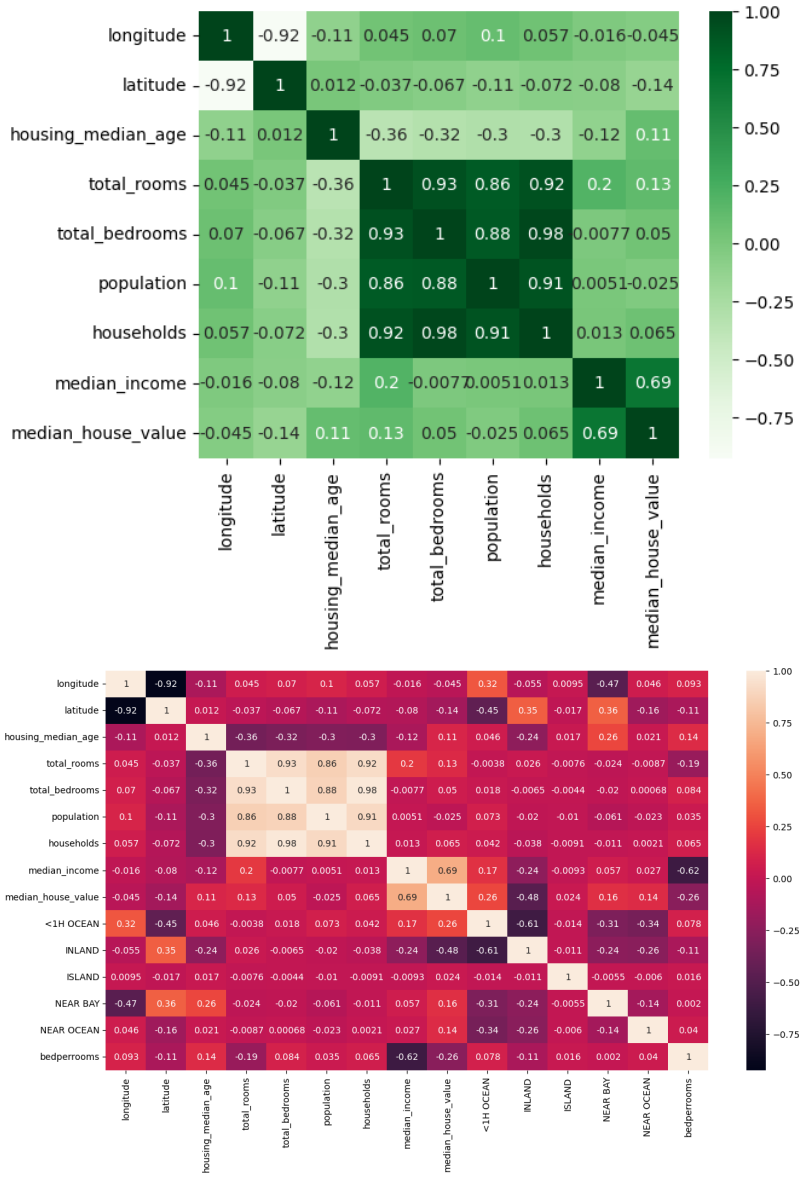


Figure 33: Correlation for California

F.5 Distribution Analysis Image

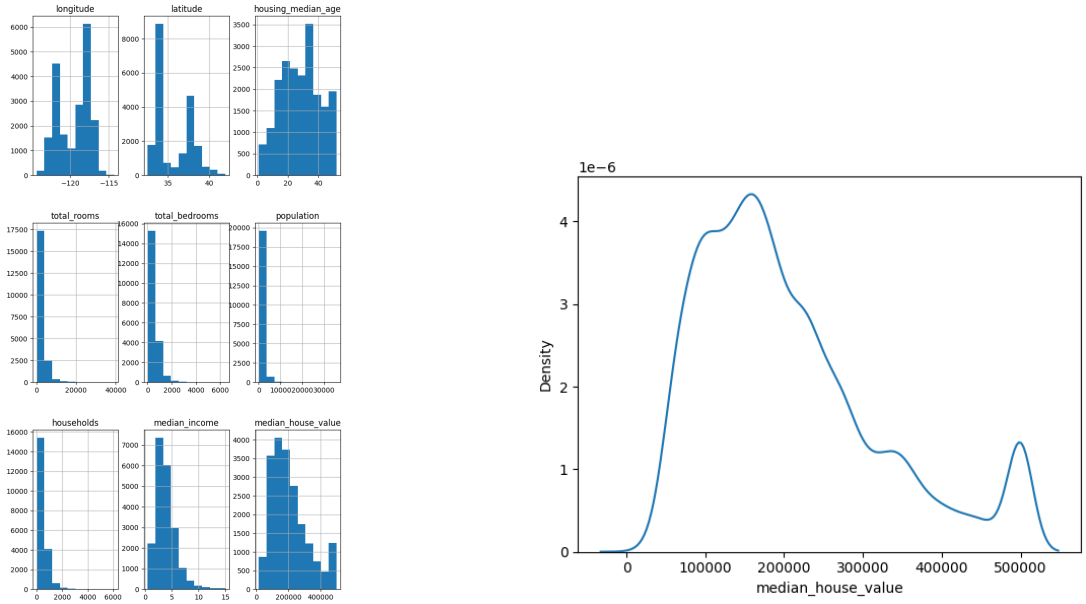


Figure 34: Features and target distribution for California dataset

F.6 Relationship between population and median_house_value image

Figure 35: Relationship between population and median_house_value

F.7 Splitting into train and test set

```
X_train, X_test, y_train, y_test =
train_test_split(X_cali,y_cali, random_state = 1501)
```

F.8 Feature scaling code

```
scaler = MinMaxScaler()
n = len(X_cali.columns)
scaler.fit(X_train.iloc[:, :n])
custom_scale = lambda X:
np.hstack((scaler.transform(X.iloc[:, :n]), X.iloc[:, n:]))
X_train_scaled = custom_scale(X_train)
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = custom_scale(X_test)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns = X_test.columns)
```

F.9 Kernelized Ridge Regression table

Kernel	Degree	$\alpha = 0.01$	$\alpha = 0.10$	$\alpha = 1.00$	$\alpha = 5.00$
linear	2	0.607603	0.607996	0.609568	0.603694
linear	3	0.607603	0.607996	0.609568	0.603694
linear	4	0.607603	0.607996	0.609568	0.603694
polynomial	2	0.638054	0.624937	0.607523	0.565495
polynomial	3	0.650808	0.635373	0.615125	0.586208
polynomial	4	0.663888	0.643528	0.621700	0.597032
rbf	2	0.653798	0.636988	0.610889	0.548818
rbf	3	0.653798	0.636988	0.610889	0.548818
rbf	4	0.653798	0.636988	0.610889	0.548818

Table 2: Kernalized ridge for California dataset

F.10 Pipeline code for california

```
pipe = Pipeline([
    ('scaler', MinMaxScaler()),
    ('regressor', KernelizedRidgeRegression(alpha=0.001,
    kernel= 'polynomial', degree= 4))
])
```

G United Kingdom

G.1 Removing Outlier code

```
def remove_outliers_zscore(dataframe, columns, threshold=3):
    for column in columns:
        z_scores = np.abs((dataframe[column]
        - dataframe[column].mean()) / dataframe[column].std())
        dataframe = dataframe[z_scores < threshold]
    return dataframe
before =dfy17.shape
print('Outlier rows removed for 2017 df are',before[0]-dfy17.shape[0])
before =dfy16.shape
dfy16 = remove_outliers_zscore(dfy16, ['Price'])
print('Outlier rows removed for 2016 df are',before[0]-dfy16.shape[0])
```

G.2 Handling Missing values code

```
print('2017',dfy17.isnull().sum())
2017 Price                0
Date of Transfer     0
Property Type        0
Old/New              0
Duration             0
Town/City            0
District             0
County               0
PPDCategory Type     0
Year                 0
dtype: int64
```

G.3 Time Series Analysis image and code

Below shows the time series analysis for UK Housing (London subset)

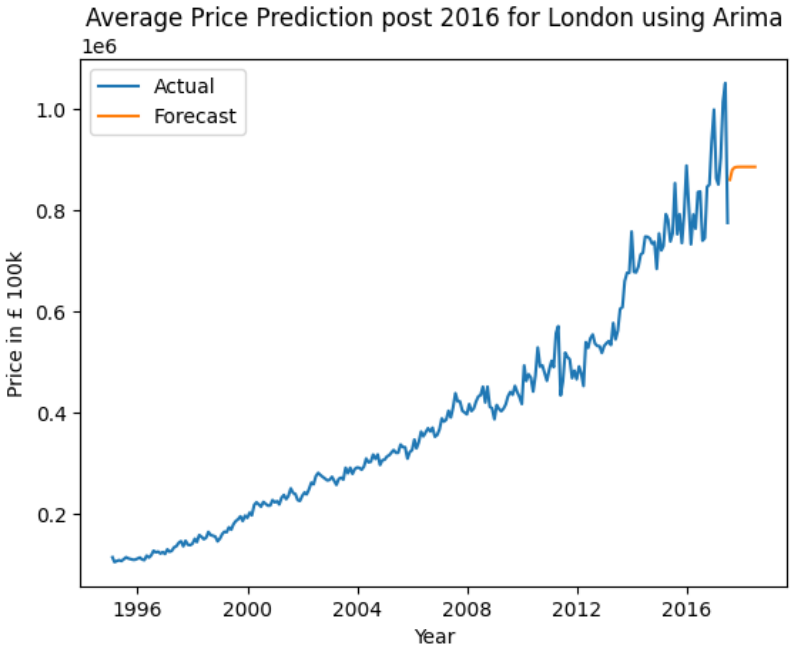


Figure 36: London time series analysis

Below is the code of the time series analysis we performed.

```
london = df [df['Town/City'] == 'LONDON']
london = london.set_index('Date_of_Transfer')
monthly_data = london['Price'].resample('M').mean()
model = ARIMA(monthly_data, order=(1, 1, 1))
results = model.fit()
forecast = results.forecast(steps=12)
```

G.4 Factorization and splitting into train and test set code

```
X['Town/City'] = X['Town/City'].factorize()[0].astype('float32')
X['District'] = X['District'].factorize()[0].astype('float32')
X['County'] = X['County'].factorize()[0].astype('float32')
X = X.drop(['Price','Date_of_Transfer','Year'],axis=1)
y = dfy17['Price']
```

G.5 New vs old towns investments image

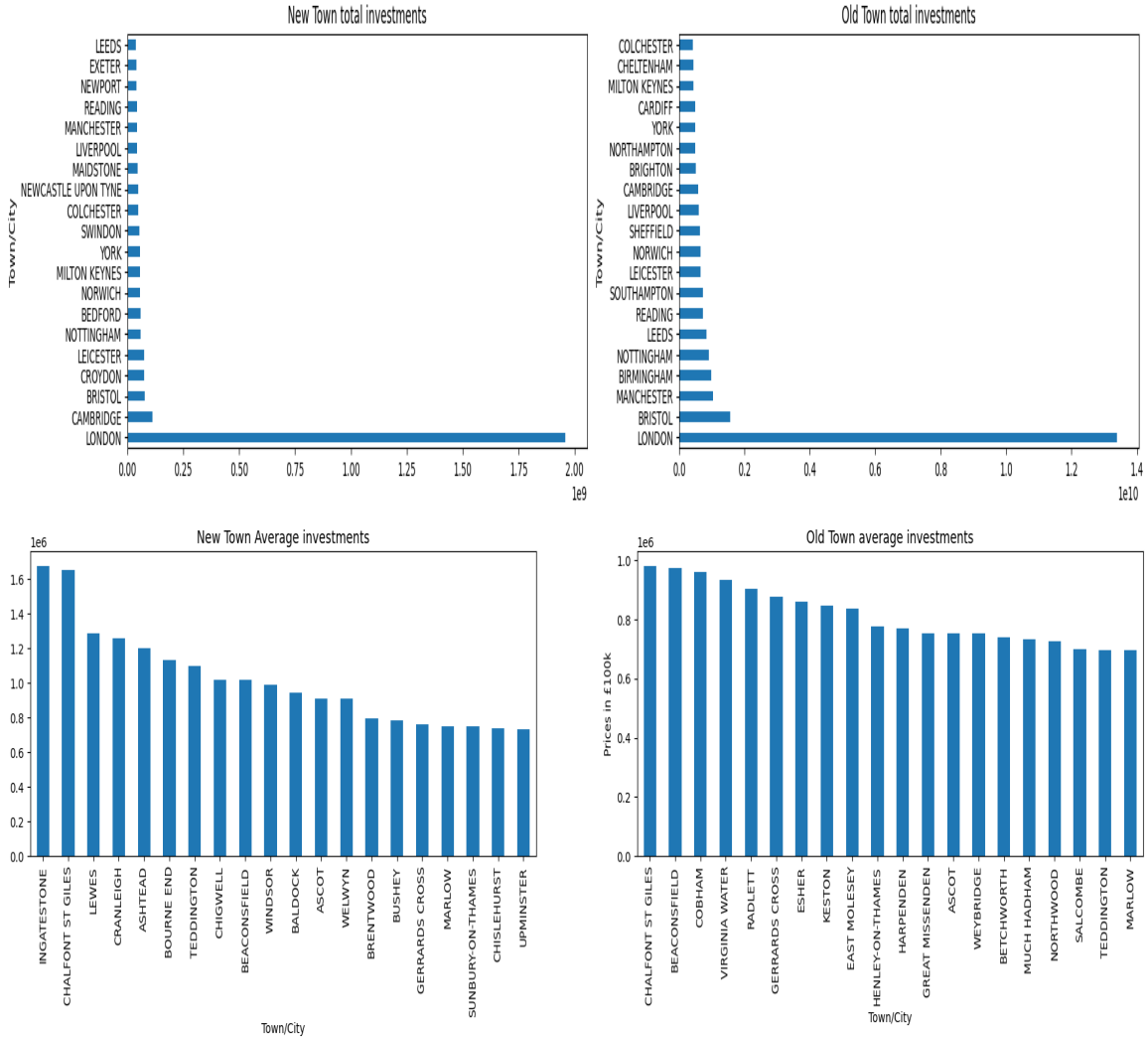
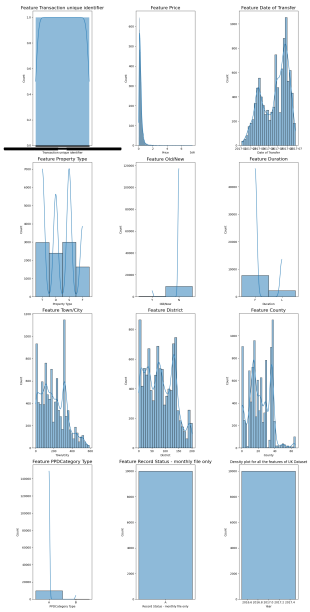


Figure 37: New vs old towns investments

G.6 Distribution Analysis Image



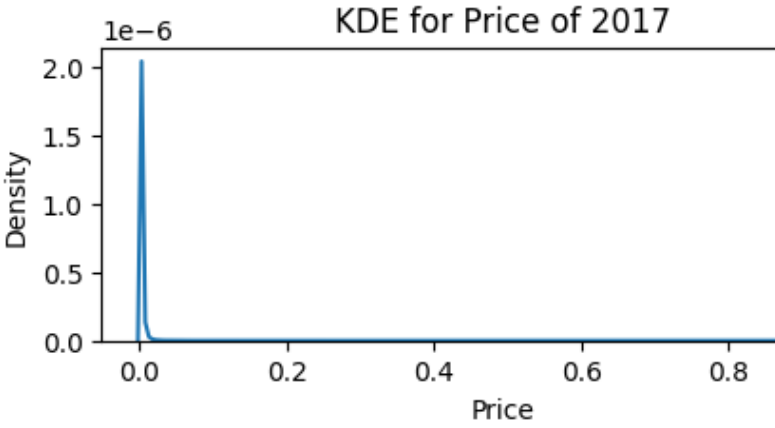
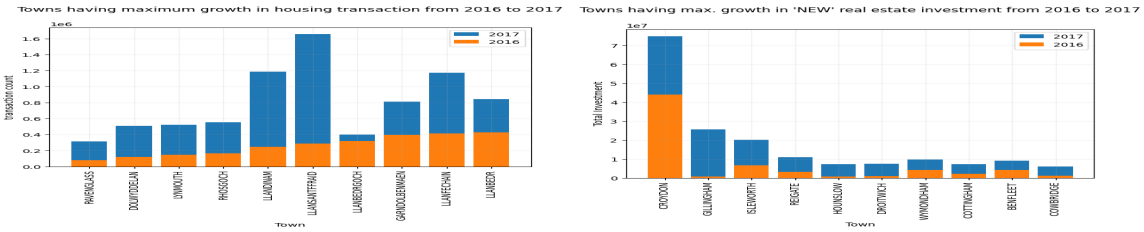


Figure 38: Features and target distribution for UK dataset

G.7 Max growth image



(a) Top 10 UK housing growth

(b) New properties growth

Figure 39: Top 10 UK housing growth and New properties growth

G.8 Growth from 2016 to 2017 image

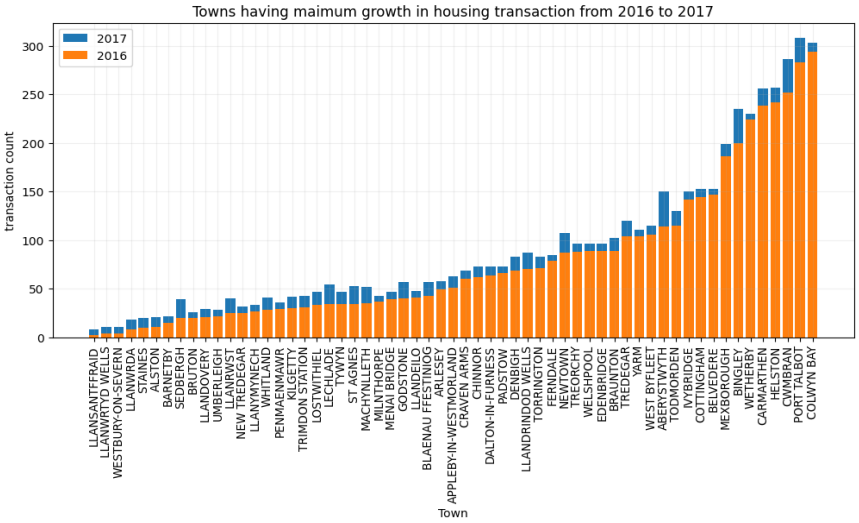


Figure 40: Uk housing growth from 2016 to 2017

G.9 Correlation analysis image

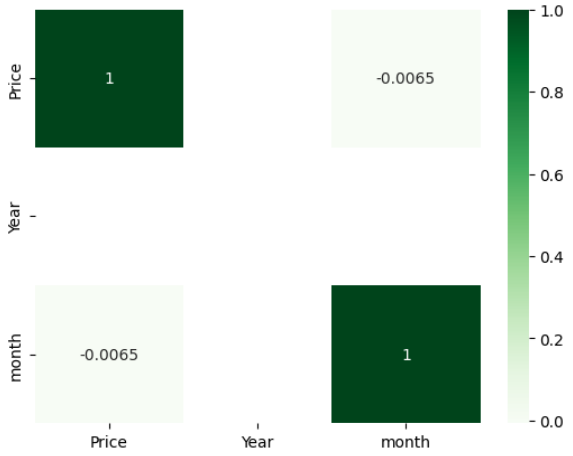
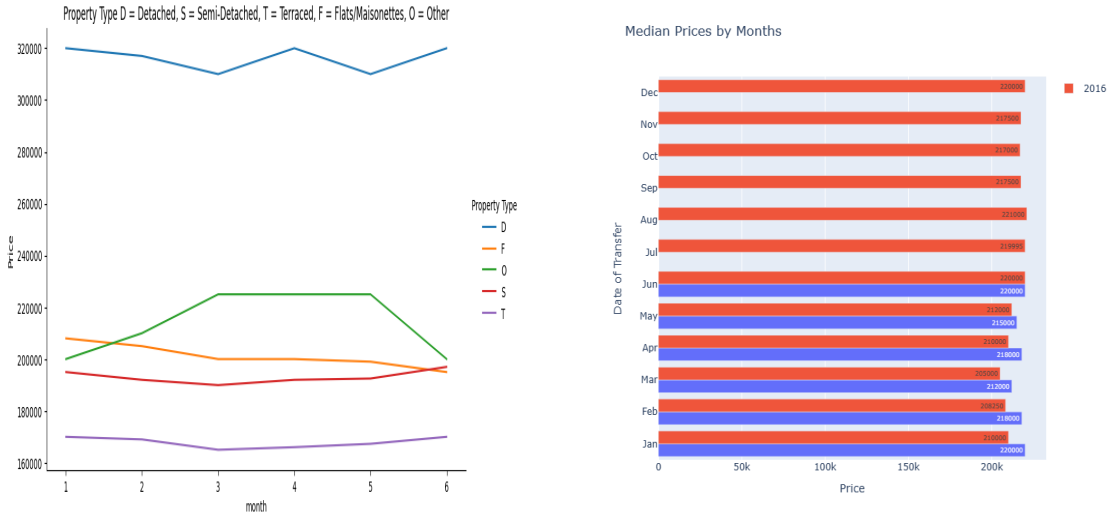


Figure 41: Correlational Analysis for United Kingdom

G.10 Property types and Price difference in individual months image



(a) Difference in property types

(b) Price difference in individual months

Figure 42: Comparison of property types and price differences in the UK

G.11 Kernelized Ridge regression table

Kernel	Degree	$\alpha = 0.1$	$\alpha = 1.0$	$\alpha = 10.0$	$\alpha = 50.0$
linear	2	0.168783	0.149626	0.129802	0.130082
linear	3	0.168785	0.149626	0.129802	0.130082
linear	4	0.168786	0.149626	0.129802	0.130082
polynomial	2	0.134735	0.132739	0.130521	0.125915
polynomial	3	0.135020	0.134285	0.132292	0.130178
polynomial	4	0.134994	0.134878	0.133531	0.132263
rbf	2	0.134926	0.133667	0.131336	0.123819
rbf	3	0.134926	0.133667	0.131336	0.123819
rbf	4	0.134926	0.133667	0.131336	0.123819

Table 3: Kernalized ridge for United kingdom dataset

G.12 Pipeline code

```
pipe = Pipeline([
    ('scaler', MinMaxScaler()),
    ('regressor', KNNRegressor(n_neighbors=2,
```

```
    distance_type = grid_search.best_params_['distance_type'],
    weighted = grid_search.best_params_['weighted']))
])
```

H Ames Housing

H.1 Data Gathering and Prepossessing

H.1.1 Loading the data

Here i have loaded the dataset in pandas dataframe and named it 'ames'.

H.1.2 Handling Duplicates

In this case since no duplicates were found we skipped removing them.

```
print(ames.duplicated().any())
```
```
False
```

H.1.3 How do we Handling Missing Values?

In this dataset there are a lot of missing values so instance so here we try to fill up the missing values by it's mean.

```
null_columns = ames.columns[ames.isnull().any()]
ames_null = ames[null_columns]
ames_null.isnull().sum()
```
```
Alley             2687
Mas Vnr Type        22
Bsmt Qual           80
Bsmt Cond           80
Bsmt Exposure       83
BsmtFin Type 1      80
BsmtFin Type 2      81
Electrical           1
Fireplace Qu      1422
Garage Type        157
Garage Finish      159
Garage Qual        159
Garage Cond        159
Pool QC           2874
Fence             2317
Misc Feature      2779
dtype: int64
```

```
ames = ames.fillna(ames.mean())
```

What is Feature Engineering? For this datasaet we have created three transformation first of log of sales price, second house age using the difference of years and finally total area using the addition first,second floor and total bsmt sf.

```
ames["LogSalePrice"] = ames['SalePrice'].apply(np.log)
ames['House Age'] = ames['Yr Sold'] - ames['Year Built']
ames['Total Area'] = ames['1st Flr SF']
+ ames['2nd Flr SF'] + ames['Total Bsmt SF']
```

H.1.4 Removing outliers

Here I'm removing any values which lie outside of two standard deviations σ.

```
\begin{lstlisting}
    def remove_outliers_zscore(dataframe, columns, threshold=3):
    for column in columns:
        z_scores = np.abs((dataframe[column]
        - dataframe[column].mean()) / dataframe[column].std())
        dataframe = dataframe[z_scores < threshold]
    return dataframe
before =ames.shape
ames = remove_outliers_zscore(ames, ['SalePrice'])
print('Outlier rows in SalePrice column in ames data are',before[0]-ames.shape
Outlier rows in SalePrice column in ames data are 45
Outlier rows removed for 2016 df are 4159
```

H.1.5 Exploratory Analysis

What is the trend of Saleprices with respect to years built? Figure 45 to explore the trend of Saleprice with respect to year build. It shows a positive linear relationship between the year the house was built and its sale price, indicating that newer houses tend to have higher sale prices. The shaded area represents the 95% confidence interval for the regression line

What are the percentage of neighbourhoods in ames? Have a look at figure 46 here we explore about the different percentage of neighbourhoods in the ames. We can see that Names has the largest percentage with 12.1% which means the highest properties are sold here followed by CollegeCr with 10.2%. This helps us understand the most trending properties within Ames dataset.

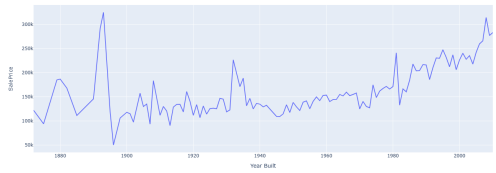


Figure 43: Trend of Saleprice vs year built

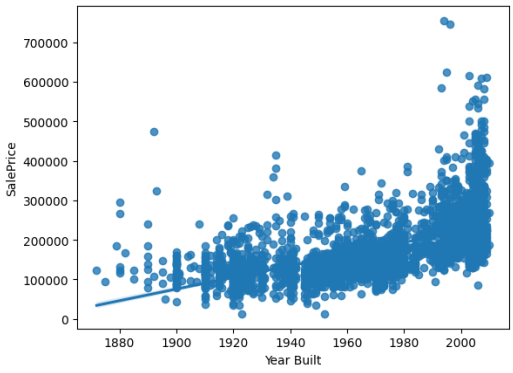


Figure 44: Trend of Saleprice vs year

Figure 45: Saleprice Trend

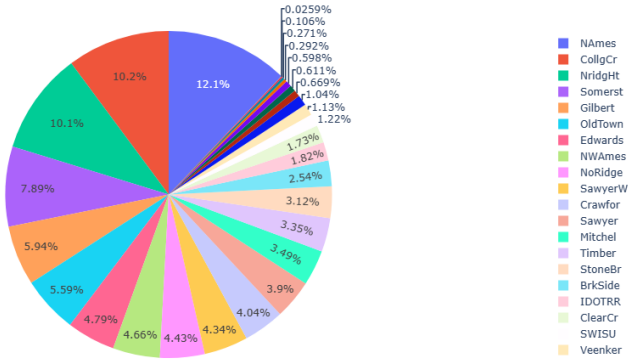


Figure 46: Percentage of neighbourhoods

What is the effect of total area and sale price to quality of living for different neighbourhoods Have a look at figure 47 shows the effect of quality of life with respect to area and price. It seems that NoRidge neighbourhood has the best quality of life of 10 with 750k sale price and 6k of total area while Edwards neighbours have the best quality of life of 10 with sale price of 175k with total area 11k.

What is the relation of Garage type, Garage area to the Sale price? In this section we will explore how garage type and Garage area affects the saleprice. Looking at figure 48. We can see from visualization that there is a positive correlation of garage area with sale price. How the outliers lie within the based on garage type based on sale price.

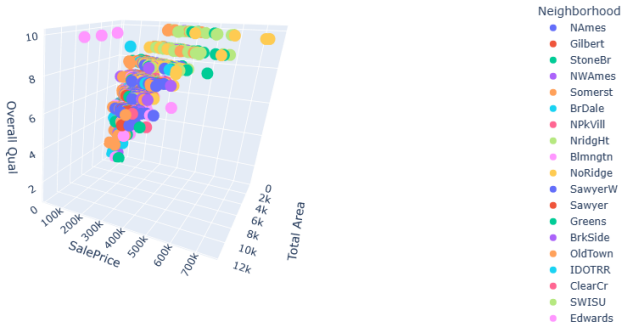


Figure 47: Effect of Saleprice and total area on quality of life for different neighbourhoods

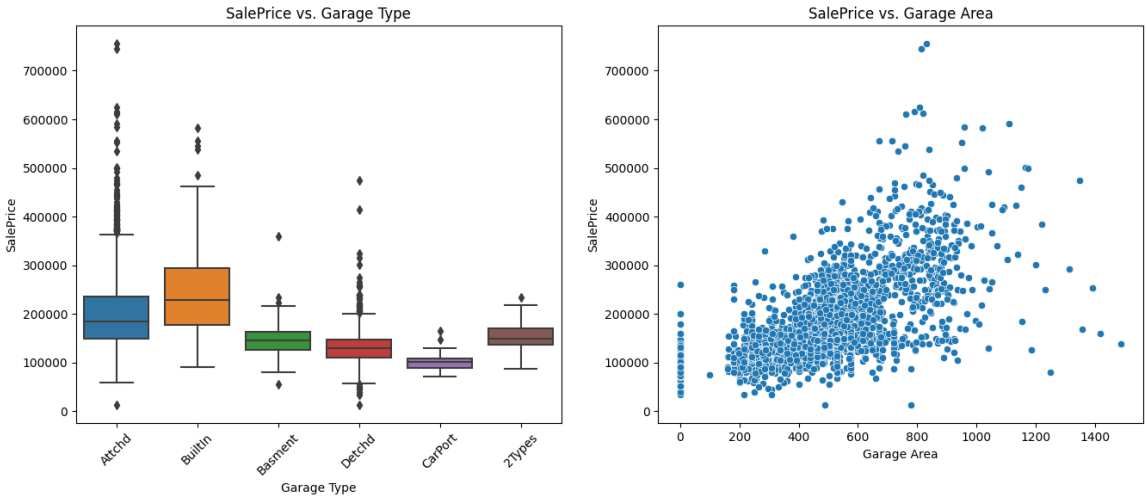


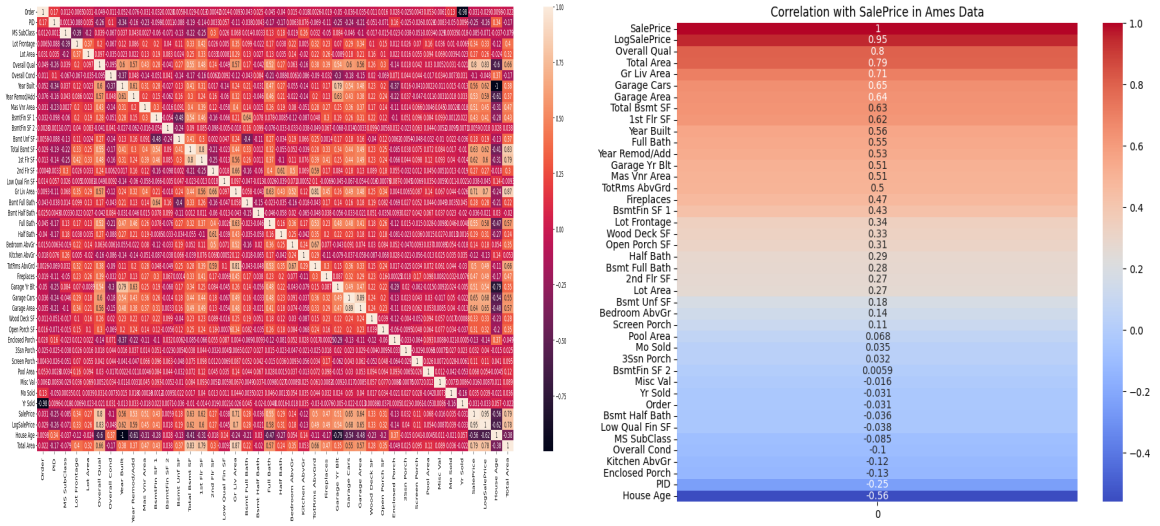
Figure 48: Garage type and area with respect to saleprice

H.2 Correlation analysis

Here we calculate the correlation matrix between features to identify which variables are positively or negatively correlated with each other. This helps to understand the inter dependencies between different features. Fig 49a shows relationship between the features all the features.

What is the correlation features among data? how is target correlated in the ames to all other features? Here i tried to plot all the numerical columns correlation among the dataset from the figure 49b we can see that that logsaleprice and saleprice are

highly corelated while Pool Area, Mo Sold, 3Ssn Porch, Misc Val, Yr Sold, BsmtFin SF 2, MISC Val, Order,Bsmt Half Bath,Screen Porch,Bedroom AbvGr, Low Qual Fin SF, MS SubClass, Overall Cond, Kitchen AbvGr, Enclosed Porch, PID shows almost no co-relation.



(a) Correlation of features in Ames

(b) correlation with target

Figure 49: Correlation plots

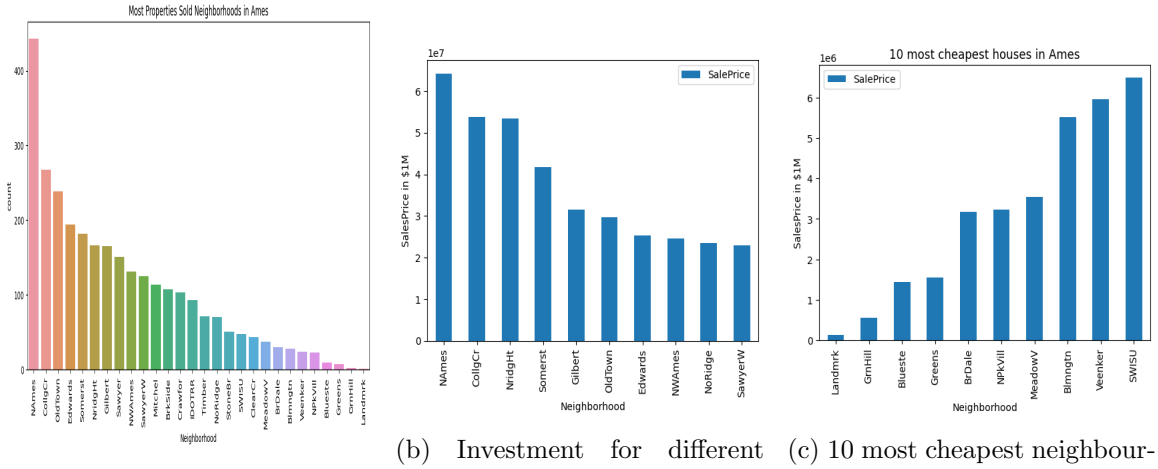
Splitting train and test set Here we split the data into train and set set before applying the algorithms.

```
X = ames.select_dtypes(['int','float']).drop('SalePrice', axis=1)
y = ames['SalePrice']
X_train, X_test, y_train, y_test =  train_test_split(X, y, random_state=1501)
```

Feature Scaling Here i have used MinMaxScaler is a technique used for feature scaling, which transforms the features to a common scale between a specified range, often between 0 and 1. The Boston Housing Dataset contains various features with different scales and magnitudes. Some features may have larger ranges than others, which can lead to certain features dominating the learning process and affecting the model's performance. Also using neural networks using above scalar allows me to converge faster and more reliably when the input features are on a similar scale.

```
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = pd.DataFrame(scaler.transform(X_train), columns = X_train.colu
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns = X_test.columns
```

H.3 Geographic Analysis



(a) Most sold houses

(b) Investment for different neighbourhoods

(c) 10 most cheapest neighbourhoods

Figure 50: Geolocation Analysis for Ames

Which are the houses with most sold neighbourhoods Figure shows the houses sold in Ames. Figure 50a that 'Names' has the most houses sold.

What is the top 10 invested neighbourhoods? Here we group the data based on different neighbourhoods and Saleprice. From figure 50c we can see has the most investment in houses.

Which are the top 10 most cheapest neighbourhoods? Here we sort the data based on saleprice and find the top 10 most cheapest sold houses as shown in figure 50c.

H.4 Distibution Analysis for Ames

From the figure 51 shown shows feature,target distribution the total area and log sale price seems normally distributed all the other columns are positively or negatively skewed hence we apply scalars to fix the issue. Looking at the target it seems to follow a Gaussian distribution.

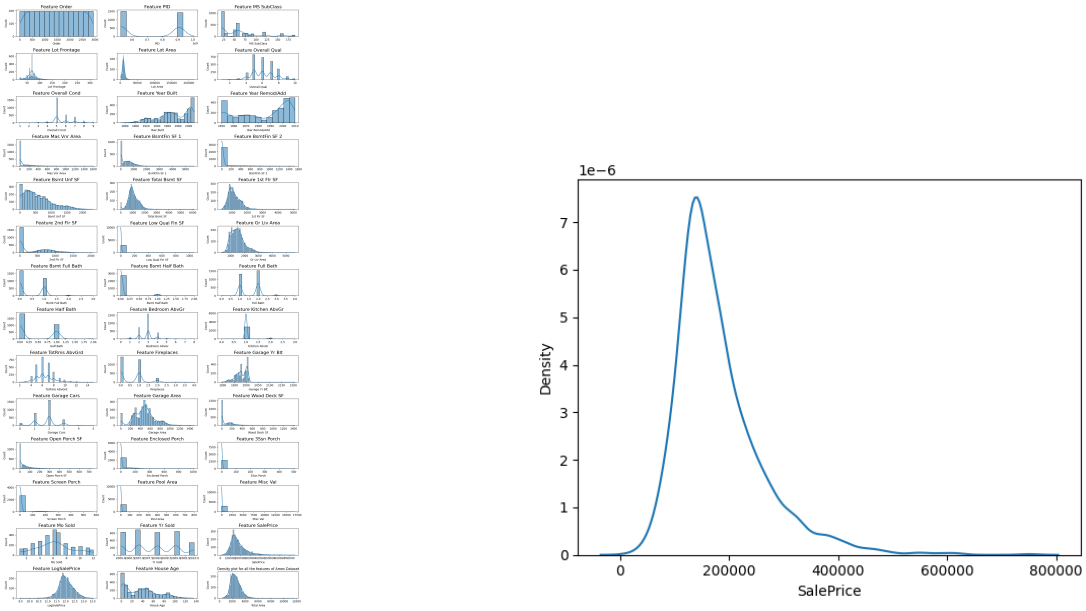


Figure 51: Features and target distribution for Ames dataset

H.5 Experimental Research

H.5.1 K-Nearest Neighbors

We apply the K-Nearest Neighbors model to the Ames Housing Dataset with a parameterized grid as follow and got these optimal results.

```
param_grid = {
    'n_neighbors': range(1, 6),
    'distance_type': [1, 2],
    'weighted': [True, False]
}
Optimal hyperparameters: {'distance_type': 1,
'n_neighbors': 5, 'weighted': True}
Best R2 Score: 0.8680010198948442
```

Further experiments With these optimal parameters we experiment with 15 nearest neighbours and see how we the training and testing R^2 vary with these params. From the fig 52 we can see that the model starts overfits the samples during 1 Nearest Neighbour then starts to evaluate other data-points in consideration and increase its complexity till it reaches to 5 Nearest Neighbours after that it starts to overfit the data. with these optimal parameters found out that its working best with 5 nearest neighbours, Manhattan distance and weighted distance

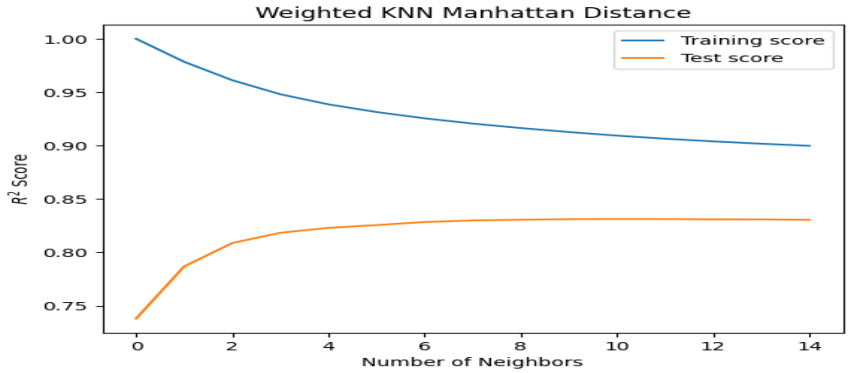


Figure 52: KNN neighbours experiment with optimal hyperparams for Ames

H.5.2 Random Forest

Here firstly we calculate the max number of features. The max features p for Ames housing dataset comes out to be 8.

$$p = int(np.ceil(X\_train.shape[1]/3))$$

Now we apply the RandomForest Regressor and get these optimal hyperparameters

```
param_grid = {
    'min_samples_split': [2, 10, 100],
    'max_depth': [5, 15, 25]
}
Best hyperparameters: {'min_samples_split': 5,
'max_depth': 10, 'n_estimators': 50}
Best R2 Score: 0.9240563826752323
```

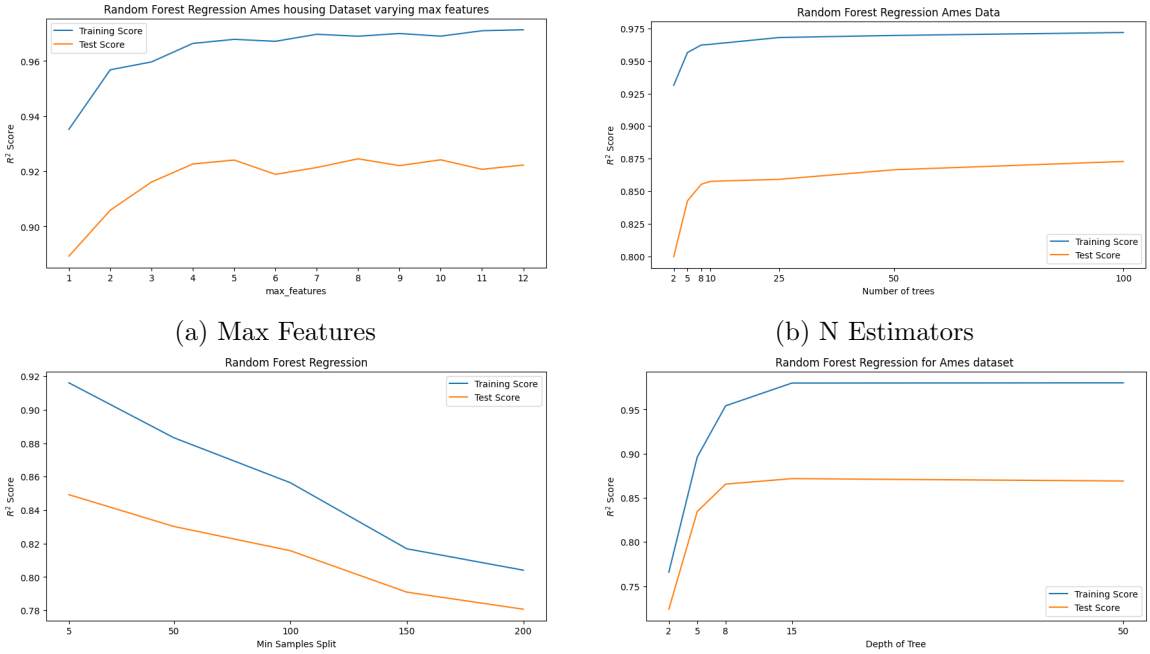
Further Experiments

Changing the max features with best hyperparameters for Ames Here we vary the max features p from 1-13 to see it's affect on training and testing scores. From fig 53a we can see that the model is over-fitting till 8 features after it remains constant.

Changing the n_estimators with best hyperparameters for Ames In this case we experiment with 1,2,5, 10,25,50,100 n_estimators. From the figure 53b we can see that the model learns till 10 n_estimators as it significantly improves testing and training scores after 10 estimators it improves slightly the data till 50 and then remains constant.

Changing the min_samples_split with best hyper-parameters Here we experiment with 1, 3, 5, 50, 100, 150, 200 of sample splits. From the figure 53c after 5 min_sample_split it keeps on underfitting the model later.

Changing the max_depth with best hyperparameters Here we experiment with depth 2, 5, 8, 15, 50 of max_depths. From figure 53d we can see that the model learns significantly by increasing the depth till it reaches 10 max_depth after that it remains constant.



(a) Max Features

(b) N Estimators

(c) Min Samples Split

(d) Max Depth

Figure 53: Hyperparameter Experiments for Ames

H.5.3 Multilayered Perceptron

To apply the Multilayered Perceptron model to the Ames Housing Dataset with activation function relu with two hidden, input layers and 1000 epochs of training

```
n_inputs = X_train.shape[1]
n_hidden1 = 30
n_hidden2 = 30
n_outputs = 1
Test MSE: 4567332805.6895
Test R-squared score: -0.0006
```

It seems that due to the nature of dataset we are unable to fit the MLP model. Since we are unable to fit the model we are dismissing the MLP model for final comparisions.

H.5.4 Kernelized Ridge Regression

On applying the Kernelized RidgeRegression model to the Ames housing dataset with a parameterized grid as follows I got these optimal results.

```
param_grid = {
    'kernel': ['linear', 'polynomial', 'rbf'],
```

```
    'alpha': [0.1, 1, 10, 50],
    'degree': [2, 3, 4]
}
Best hyperparameters: {'alpha': 0.1,
'kernel': 'polynomial', 'degree': 4}
Best $R^2$ Score: 0.8589
```

Now we save the testing scores of each above parameters and figure 54 shows the results of three separate groups of kernels. The score results for each are shown in table 4.

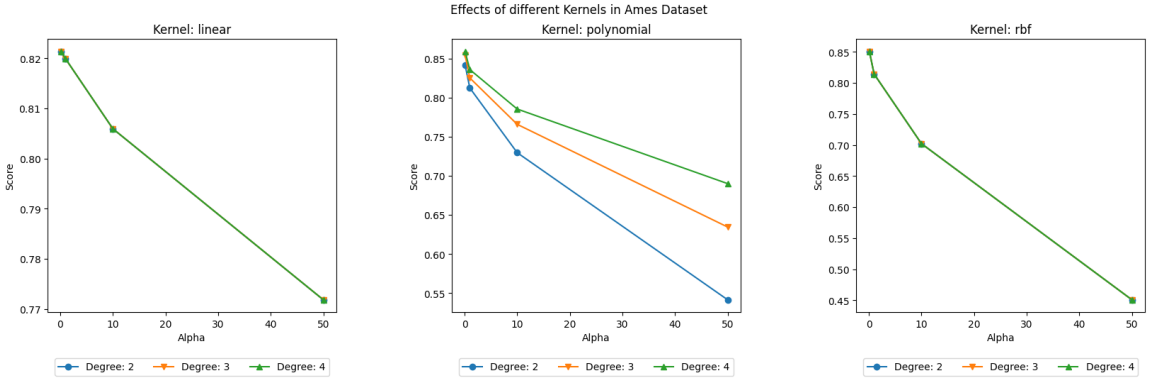


Figure 54: Test R^2 scores for different kernels for Ames

Kernel	Degree	$\alpha = 0.1$	$\alpha = 1.0$	$\alpha = 10.0$	$\alpha = 50.0$
linear	2	0.821375	0.819913	0.805951	0.771853
linear	3	0.821375	0.819913	0.805951	0.771853
linear	4	0.821375	0.819913	0.805951	0.771853
polynomial	2	0.841540	0.812191	0.729542	0.541508
polynomial	3	0.854046	0.824997	0.765743	0.634477
polynomial	4	0.858920	0.835729	0.785322	0.689969
rbf	2	0.850813	0.814370	0.702219	0.450735
rbf	3	0.850813	0.814370	0.702219	0.450735
rbf	4	0.850813	0.814370	0.702219	0.450735

Table 4: Kernalized ridge for Ames housing dataset

H.5.5 Final Comparison

In this part we compare different models performance the best R^2 scores achieved on the Ames housing dataset. Figure 55 shows that K-Nearest neighbour outperforms all the other algorithms.

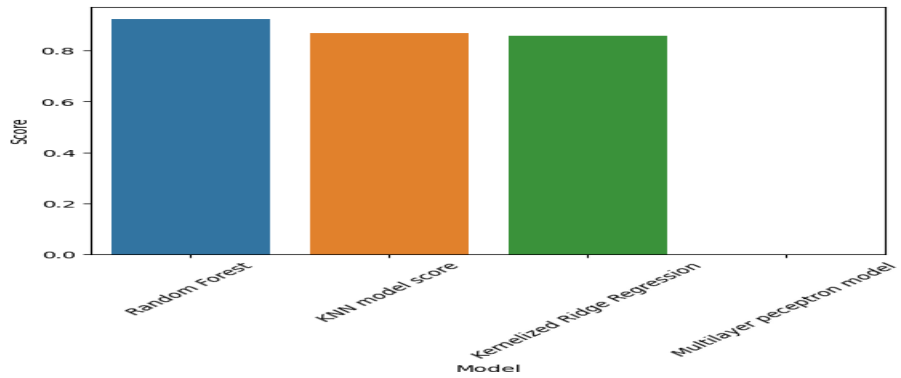


Figure 55: Comparing different models R^2 scores) for Ames

H.5.6 Ames Pipeline

We apply our best models with best hyper-parameters of Ames housing dataset and build our Kernelized Ridge regression pipeline.

```
pipe = Pipeline([
    ('scaler', MinMaxScaler()),
    ('regressor', RandomForestRegressor(n_estimators=50,
    max_features=8,min_samples_split=5, max_depth=10))
])
R2 score for the pipeline: 0.8701153470251737
```

H.5.7 Inductive Conformal validity for Ames

From visualization of figure 28c we can see that the error rate and average with are inversely proportional and at 5% significance we have maximum error rate and lowest average width on increasing significance level ϵ we see error rate increase and width decrease which provides a validity for the above pipeline to be under inductive conformal.

I Inductive Conductive Conformal code

Below is the code for Inductive conformal prediction.

```
class InductiveConformalRegressor:
    def __init__(self, model):
        self.model = model

    def fit(self, X_train, y_train, X_cal, y_cal):
        # Fit underlying model to proper training set
        self.model.fit(X_train, y_train)

        # Calculate nonconformity scores for calibration set
        y_pred_cal = self.model.predict(X_cal)
        self.nc_scores_ = np.abs(y_pred_cal - y_cal)

    def predict(self, X, significance=0.05):
        # Make predictions using underlying model
        y_pred = self.model.predict(X)

        # Calculate prediction intervals
        n = len(self.nc_scores_)
        k = int(np.ceil(significance * n))
        threshold = np.partition(self.nc_scores_, -k)[-k]
        interval = np.abs(y_pred - threshold)
        lower = y_pred - interval
        upper = y_pred + interval
        return lower, upper

icp = InductiveConformalRegressor(pipe)
icp.fit(X_train, y_train, X_cal, y_cal)
significance_levels = np.linspace(0, 1, 101)
coverage = []
avg_width = []
for significance in significance_levels:
    lower, upper = icp.predict(X_test.select_dtypes
    (include=['int64', 'float64']), significance=significance)
    in_interval=np.logical_and(lower <= y_test, y_test <= upper)
    coverage.append(np.mean(in_interval))
    width = upper - lower
    avg_width.append(np.mean(width))
```

J How to run my code

All the code utilized in this project can be found within the submitted `QuidProject.zip` folder. A brief summary of the files are given below.

- `KNN.py` for K-Nearest Neighbors Regression Model
- `RFR.py` for Random Forest Regression Model
- `RR.py` for Ridge Regression Model
- `KernelizedRR.py` for Kernelized Ridge Regression Model
- `MLP.py` for Multilayer Perceptron Model

Additionally, notebooks containing the analysis for different analysis for each datasets can be found under the filenames:

- `Boston_Housing_Dataset_Analysis.ipynb`
- `California_Housing_Dataset_Analysis.ipynb`
- `Ames_Housing_Dataset_Analysis.ipynb`
- `UK_Housing_Dataset_Analysis.ipynb`

First step is extract the project then download the dataset uk housing dataset from kaggle `https://www.kaggle.com/datasets/hm-land-registry/uk-housing-prices-paid` and place it in the Datasets folder. Then upload the folder on google drive. Once you upload the folder. To view the analysis, simply open the notebooks on google drive and execute the cells sequentially. Note that some cells will cause long runtimes Therefore, exercise caution before re-running algorithms fitting cells.

If you want to run code the code locally all the dependencies, including NumPy, pandas, matplotlib, seaborn, and scikit-learn, plotly are specified in the `requirements.txt` file. for running in your local system install these dependencies using the command `pip install -r requirements.txt` and comment out libraries for colab mounting in the import while running locally.

I would strongly recommend uploading all the files on google drive and running it on colab as there would be no installation required as all of the libraries are preexisting in colab.

J.1 Python Version

This project was built using Python 3.10.12 of colab. It is recommended to use Python 3 or a higher version for successful code execution.

K KNNRegressor

```
class KNNRegressor:
    def __init__(self, n_neighbors = 3, distance_type =2 , weighted = False):
        self.n_neighbors = n_neighbors
        self.distance_type = distance_type
        self.weighted = weighted
        print(f'KNNRegressor(n_neighbors={n_neighbors},
        distance_type={distance_type}, weighted={weighted})')

    def fit(self, X_train, y_train):
        if type(X_train) != np.ndarray:
            X_train = X_train.to_numpy()
            y_train = y_train.to_numpy()
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        if type(X_test) != np.ndarray:
            X_test = X_test.to_numpy()
        y_preds = [self._get_prediction(x) for x in X_test]
        return np.array(y_preds)

    def _get_prediction(self, x):
        # Compute the distances between x and all examples in the training set
        if self.distance_type == 1:
            distances = [np.sum(np.abs(x - x_train)) for x_train in self.X_trai
        else:
            distances = [np.sqrt(np.sum((x - x_train) ** 2)) for x_train in se

        # Getting the indices of k mininum distances (nearest neighbors)
        k_min_indices = np.argsort(distances)[:self.n_neighbors]

        # Getting the K nearest labels from the training labels
        k_nearest_labels = [self.y_train[l] for l in k_min_indices]

        # Weighted K Nearest Neighbors Regression
        if self.weighted:
            weights = np.arange(self.n_neighbors, 0, -1)
            weights = weights/np.sum(weights)
            return np.dot(weights, k_nearest_labels)
        return np.mean(k_nearest_labels)
```

```
    def score(self, X_test, y_test):
        if type(X_test) != np.ndarray:
            X_test = X_test.to_numpy()
            y_test = y_test.to_numpy()
        y_preds = self.predict(X_test)
        rss = np.sum((y_preds - y_test)**2)
        avg_label = np.mean(y_test)
        tss = np.sum((y_test - avg_label)**2)
        r2 = (tss - rss)/tss
        return r2
    def get_params(self, deep=True):
        return {
            'n_neighbors': self.n_neighbors,
            'distance_type': self.distance_type,
            'weighted': self.weighted
        }
    def set_params(self, **params):
      for key, value in params.items():
          setattr(self, key, value)
      return self
```

L MLPRegressor

```
class MLP:
    def __init__(self, n_inputs, n_hidden1, n_hidden2,
    n_outputs, activation='sigmoid'):
        # Initialize the weights and biases of the MLP
        np.random.seed(42)
        self.weights1 = np.random.randn(n_inputs, n_hidden1)
        self.biases1 = np.zeros(n_hidden1)
        self.weights2 = np.random.randn(n_hidden1, n_hidden2)
        self.biases2 = np.zeros(n_hidden2)
        self.weights3 = np.random.randn(n_hidden2, n_outputs)
        self.biases3 = np.zeros(n_outputs)

        # Set the activation function
        if activation == 'sigmoid':
            self.activation = self.sigmoid
            self.d_activation = self.d_sigmoid
        elif activation == 'relu':
```

```
            self.activation = self.relu
            self.d_activation = self.d_relu
        else:
            raise ValueError(f'Invalid activation function: {activation}')

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def d_sigmoid(self, z):
        return self.sigmoid(z) * (1 - self.sigmoid(z))

    def relu(self, z):
        return np.maximum(0, z)

    def d_relu(self, z):
        return (z > 0).astype(float)

    def mse_loss(self, y_true, y_pred):
        y_pred = y_pred.reshape(y_true.shape)
        return np.mean((y_true - y_pred) ** 2)

    def d_mse_loss(self, y_true, y_pred):
        return 2 * (y_pred - y_true.reshape(-1, 1)) / len(y_true)

    def forward_pass(self, X):
        # Computing the predictions of the MLP on the input data
        z1 = X.dot(self.weights1) + self.biases1
        a1 = self.activation(z1)
        z2 = a1.dot(self.weights2) + self.biases2
        a2 = self.activation(z2)
        z3 = a2.dot(self.weights3) + self.biases3
        y_pred = z3
        return y_pred, z1, a1, z2, a2

    def backward_pass(self, X, y_true, y_pred, z1, a1, z2, a2):
        # Computing the new gradients of the loss
        d_z3 = self.d_mse_loss(y_true, y_pred)
        d_a2 = d_z3.dot(self.weights3.T)
        d_z2 = d_a2 * self.d_activation(z2)
        d_weights2 = a1.T.dot(d_z2)
        d_biases2 = d_z2.sum(axis=0)
        d_a1 = d_z2.dot(self.weights2.T)
```

```
        d_z1 = d_a1 * self.d_activation(z1)
        d_weights1 = X.T.dot(d_z1)
        d_biases1 = d_z1.sum(axis=0)
        d_weights3 = a2.T.dot(d_z3)
        d_biases3 = d_z3.sum(axis=0)
        return d_z3, d_weights1, d_biases1, d_weights2, d_biases2, d_weights3,

    def update_weights_biases(self, learning_rate,d_weights1, d_biases1,
    d_weights2, d_biases2, d_weights3, d_biases3):
        # Updating the weights and biases of the MLP using gradient descent
        self.weights3 =self.weights3 - learning_rate * d_weights3
        self.biases3 =self.biases3 - learning_rate * d_biases3
        self.weights2 = self.weights2 - learning_rate * d_weights2
        self.biases2 = self.biases2 - learning_rate * d_biases2
        self.weights1 =self.weights1 - learning_rate * d_weights1
        self.biases1 =self.biases1 - learning_rate * d_biases1

    def fit(self, X_train, y_train, learning_rate=0.01, n_epochs=1000):
        # Training the MLP for a fixed number of epochs
        for epoch in range(n_epochs):
            # Forward pass: compute the predictions of the MLP on the training
            y_pred, z1, a1, z2, a2 = self.forward_pass(X_train)

            # Computing the training loss
            loss = self.mse_loss(y_train, y_pred)
            # Backward pass: compute the gradients and update the weights
            and biases of the MLP
            d_z3, d_weights1, d_biases1, d_weights2, d_biases2, d_weights3,
            d_biases3 = self.backward_pass(X_train, y_train, y_pred, z1, a1, z2
            if epoch % 50 ==0:
              print(f'epochs = {epoch} ==========> loss {loss} ')
            self.update_weights_biases(learning_rate, d_weights1, d_biases1,
            d_weights2, d_biases2, d_weights3, d_biases3)

    def predict(self, X_test):
        # Compute and return the predictions of the MLP on the test set
        y_pred_test, z1, a1, z2, a2 = self.forward_pass(X_test)
        return y_pred_test
```

M RandomForestRegressor

```
class RandomForestRegressor:
    def __init__(self, n_estimators = 10, min_samples_split = 2,
                 max_depth = 50, max_features = None):
        self.n_estimators = n_estimators
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.max_features = max_features
        self.trees = []

    def fit(self, X_train, y_train):
        if type(X_train) != np.ndarray:
            X_train = X_train.to_numpy()
            y_train = y_train.to_numpy()
        self.trees = []
        for i in range(self.n_estimators):
            tree = DecisionTreeRegressor(min_samples_split =
                                         self.min_samples_split,
                                         max_depth = self.max_depth,
                                         max_features=self.max_features)
            X_bootstrapped, y_bootstrapped =
            get_bootstrapped_dataset(X_train, y_train)
            tree.fit(X_bootstrapped, y_bootstrapped)
            self.trees.append(tree)

    def predict(self, X_test):
        if type(X_test) != np.ndarray:
            X_test = X_test.to_numpy()
        tree_preds = np.array([tree.predict(X_test) for tree in self.trees])
        #print(tree_preds.shape)
        y_preds = np.array([np.mean(tree_pred) for tree_pred in
        np.transpose(tree_preds)])
        #print(y_preds.shape)
        return y_preds

    def score_r2(self, X_test, y_test):
        if type(X_test) != np.ndarray:
            X_test = X_test.to_numpy()
            y_test = y_test.to_numpy()
        y_preds = self.predict(X_test)
        rss = np.sum((y_preds - y_test)**2)
        avg_label = np.mean(y_test)
        tss = np.sum((y_test - avg_label)**2)
```

```
        r2 = (tss - rss)/tss
        return r2
```

N Kernelized Ridge Regressor

```
class KernelizedRidgeRegression:
    def __init__(self, alpha = 1, kernel = 'linear'
    , gamma = None, degree = 3, coef0 = 1):
        self.alpha = alpha
        self.kernel = kernel
        self.degree = degree
        self.coef0 = coef0
        self.optimal_r = None
        self.gamma = gamma

    def fit(self, X_train, y_train):
        if self.gamma == None:
            self.gamma = 1.0/X_train.shape[1]
        self.X = X_train
        l2_penalty = self.alpha * np.identity(self.X.shape[0])
        self.optimal_r = y_train.T @ np.linalg.inv(self.kernel_pred
        (self.X, self.X.T) + l2_penalty)

    def kernel_pred(self, A, B):
        if self.kernel == 'linear':
            return A @ B
        elif self.kernel == 'polynomial':
            return ((self.gamma * (A @ B)) + self.coef0) ** self.degree
        elif self.kernel == 'rbf':
            if type(A) != np.ndarray:
                A = A.to_numpy()
            if type(B) != np.ndarray:
                B = B.to_numpy()
            A_norm = np.sum(A ** 2, axis = -1)
            B_norm = np.sum(B ** 2, axis = -1)
            if A.shape[0] != B.shape[0] and A.shape[1] == B.shape[1]:
                return np.exp(-self.gamma *
                (A_norm[:, None] + B_norm[None, :] - 2 * np.dot(A,B.T)))
            else:
                return np.exp(-self.gamma * (A_norm[None, :] + A_norm[:, None]
                - 2 * np.dot(A,A.T)))
```

```
def predict(self, X_test):
    if self.kernel == 'rbf':
        y_preds = self.optimal_r @ self.kernel_pred(self.X, X_test)
    else:
        y_preds = self.optimal_r @ self.kernel_pred(self.X, X_test.T)
    return y_preds

def score(self, X_test, y_test):
    if type(X_test) != np.ndarray:
        X_test = X_test.to_numpy()
        y_test = y_test.to_numpy()
    y_preds = self.predict(X_test)
    rss = np.sum((y_preds - y_test)**2)
    avg_label = np.mean(y_test)
    tss = np.sum((y_test - avg_label)**2)
    r2 = (tss - rss)/tss
    return r2
```