BUDT758T: Enterprise Cloud Computing and Big Data



# *House Prices - Advanced Regression Techniques*

***Kaggle Competition***

*By Group 17*

| Aagney Menon | 120433573 |
| --- | --- |
| Harish Varma Bhupahiraju | 120429620 |
| Solayappan Ganesaan | 120095870 |
| Alireza Bavafa | 120380273 |
| Akshay Sharma | 119542859 |

## Team Name: Team\_Project\_17

## Original code example used

<https://www.kaggle.com/code/gusthema/house-prices-prediction-using-tfdf>

<https://www.kaggle.com/code/smitzaveri/linear-regression-house-prices-prediction>

# Methods Used

### Handling Missing Values

train[train.\_get\_numeric\_data().columns].isnull().sum()

train.info()

**Identification**

The first step is to identify missing values in the dataset. This was done using methods like isnull() or info() to check for null values in each column.

**Imputation**

Missing values are then filled in using appropriate strategies.

cols = ['BsmtFinSF1', 'BsmtFinSF2', 'BsmtFullBath', 'BsmtHalfBath', 'GarageArea',

'GarageCars', 'MasVnrArea', 'TotalBsmtSF', 'GarageYrBlt', 'BsmtUnfSF']

df[cols] = df[cols].fillna(value=0)

For numeric features like BsmtFinSF1, BsmtFinSF2, BsmtFullBath, etc., missing values are filled with a default value of 0. This strategy assumes that missing values indicate the absence of a certain feature, such as the absence of a finished basement or a full bathroom. Similarly, missing values in the GarageYrBlt feature, representing the year the garage was built, are filled with a default value of 0, assuming that missing values indicate the absence of a garage.

df['LotFrontage'] = df.groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.fillna(x.median()))

For the LotFrontage feature, missing values are filled based on the median of each neighborhood. This approach leverages the relationship between LotFrontage and Neighborhood to impute missing values more accurately, considering neighborhood-specific characteristics.

df['Functional'] = df['Functional'].fillna('Typ') # Fill missing values in 'Functional' column with 'Typ'

df['Electrical'] = df['Electrical'].fillna("SBrkr") # Fill missing values in 'Electrical' column with 'SBrkr'

For categorical features like Functional, Electrical, KitchenQual, etc., missing values are filled with default values such as 'Typ' (for 'Functional'), 'SBrkr' (for 'Electrical'), 'TA' (for 'KitchenQual'), etc. These default values are chosen based on domain knowledge or the most frequent value in the respective column.

cols = ['BsmtQual', 'BsmtCond', 'FireplaceQu', 'GarageQual']

df[cols] = df[cols].fillna(value='NA')

Quality-related columns like BsmtQual, FireplaceQu, GarageQual, etc., where missing values may indicate the absence of a certain quality feature, are filled with 'NA' to signify that the feature is not applicable or not available.

df['MSZoning'] = df.groupby('MSSubClass')['MSZoning'].transform(lambda x: x.fillna(x.mode()[0]))

The MSZoning feature, representing the zoning classification of the property, is filled based on the mode of each MSSubClass. This strategy leverages the relationship between MSZoning and MSSubClass to impute missing values more accurately, considering property subclass-specific zoning classifications.

Quality-related columns: Special treatment is given to quality-related columns by replacing missing values with "NA", indicating that the feature is not applicable or not available.

Any remaining columns with missing values are dropped from the dataset using the .dropna() method. This ensures that the dataset used for model training does not contain missing values, which could otherwise lead to errors during model fitting.

**Encoding Categorical Variables**

**Ordinal Encoding**

ordinal\_columns = ordinal\_encoder.fit\_transform(df[cols])

Ordinal encoding is applied to categorical variables where there is a clear order or hierarchy among the categories. This encoding preserves the ordinal relationship between categories.

Ordinal encoding is primarily used in our code for quality-related features such as ExterQual, ExterCond, BsmtQual, BsmtCond. The order of categories for each ordinal variable is predefined, typically based on domain knowledge. For example, the quality-related features are encoded with categories like 'NA', 'Po' (Poor), 'Fa' (Fair), 'TA' (Average/Typical), 'Gd' (Good), and 'Ex' (Excellent).The OrdinalEncoder from scikit-learn is used to perform ordinal encoding. This encoder accepts the predefined order of categories and assigns numerical values accordingly.

**Nominal Encoding (One-Hot Encoding)**

encoder = OneHotEncoder(handle\_unknown='ignore', sparse\_output=False)

Nominal encoding, also known as one-hot encoding, is used for categorical variables where there is no inherent order among the categories. It creates binary features for each category, with a value of 1 indicating the presence of that category and 0 indicating absence.

In our code, nominal encoding is applied to nominal categorical variables such as MSZoning, Street, Alley, and LotShape. The OneHotEncoder from scikit-learn is used to perform one-hot encoding. It creates binary features for each category in the nominal variables, effectively expanding the feature space. After one-hot encoding, the original categorical variables are replaced with the binary features representing each category. This results in a sparse matrix where most values are zeros, except for the presence of each category.

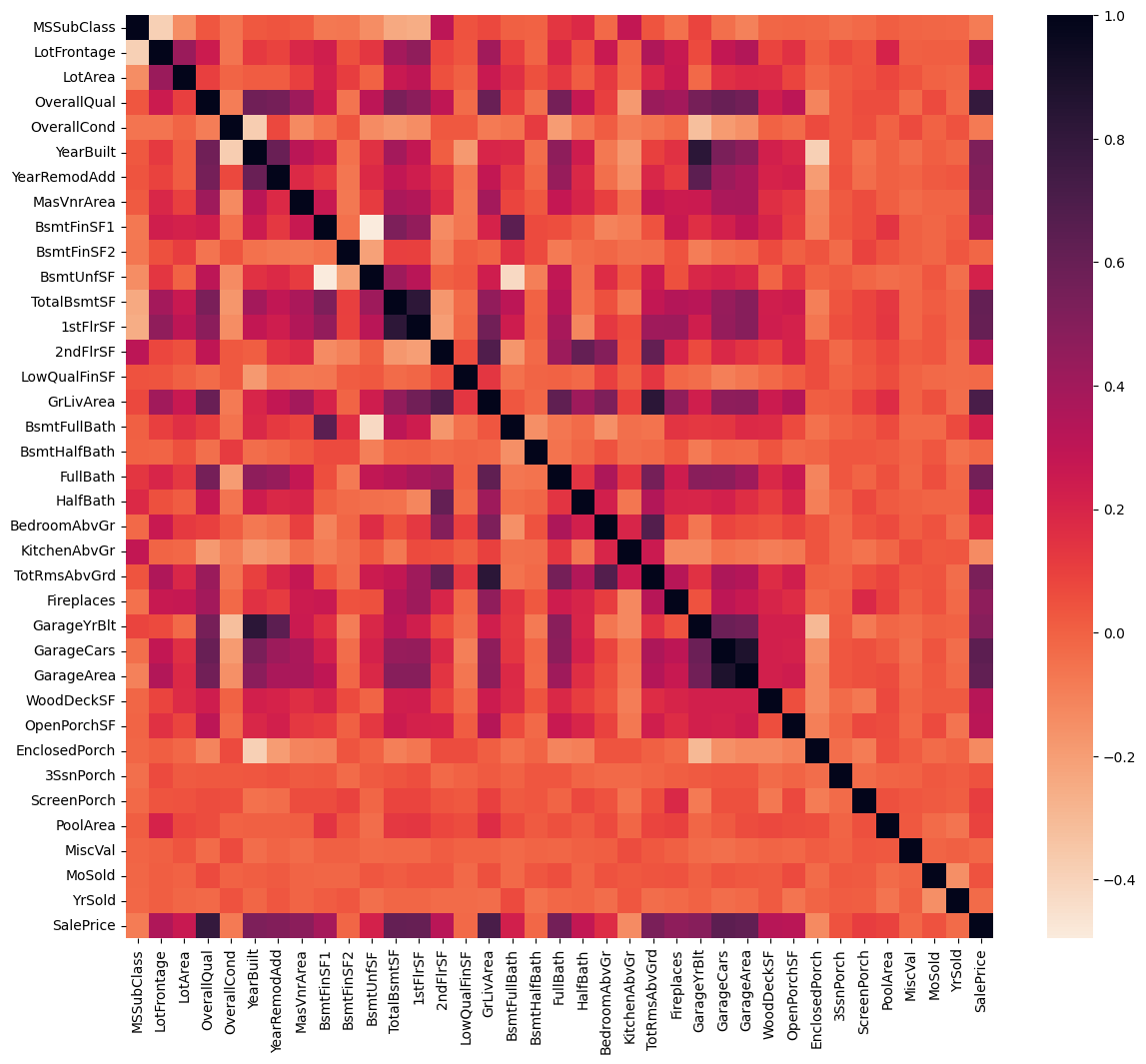
**Data Splitting**

Before splitting the data, the target variable (SalePrice) was separated from the feature variables. This ensured that the target variable is not included in the features used for model training and evaluation.

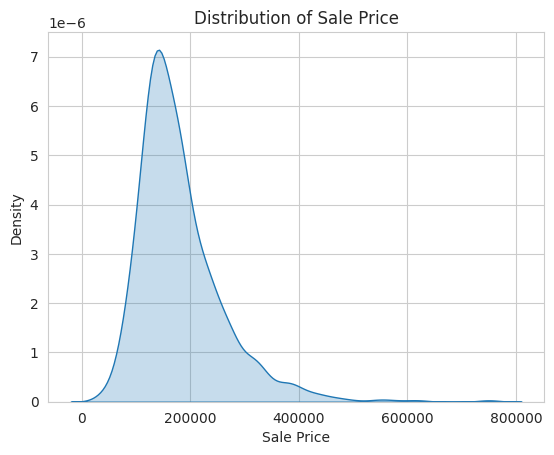
The train\_test\_split function from scikit-learn is used to split the dataset into training and testing sets, randomly shuffling the data and splitting it into two portions based on the specified ratio.

The parameter test\_size=0.2 specifies the proportion of the dataset to include in the testing set to 20%, meaning 80% of the data will be used for training. The parameter random\_state was set to 43 to ensure reproducibility of the data. Shuffling was set to true because it helps ensure that the data points are randomly distributed between the training and testing sets, reducing bias.

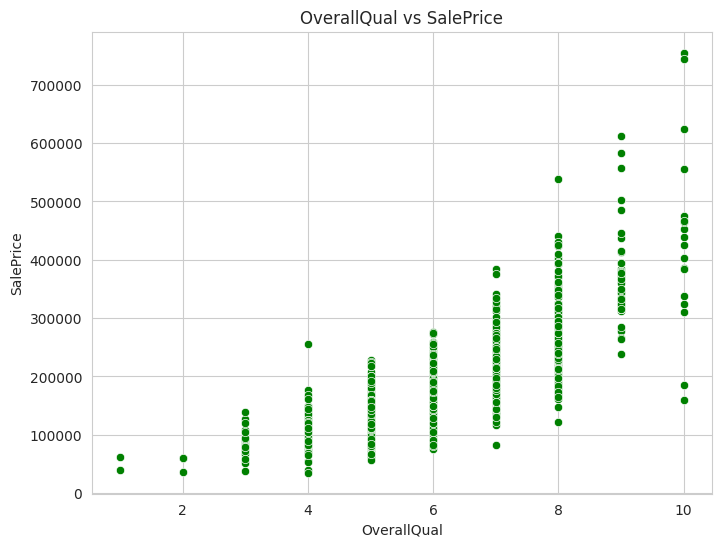
### Exploratory Data Analysis

Correlation Matrix: 

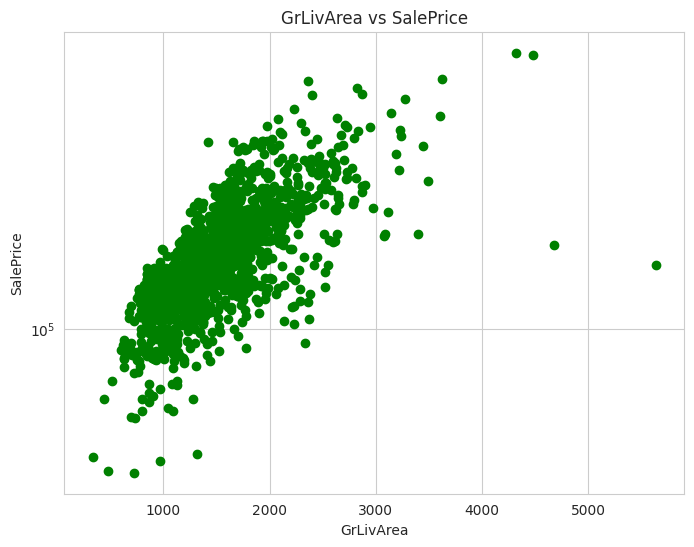
The correlation matrix shows the correlation between the variables and the target variable. In this case, the central variables (in the square from the top left corner to GarageArea (as the lower left corner) and OpenPorchSF (lower right corner) can be marked as the zone with the most correlated variables. However, there is not a definitive trend and the more important variables were simply obtained via the output table.



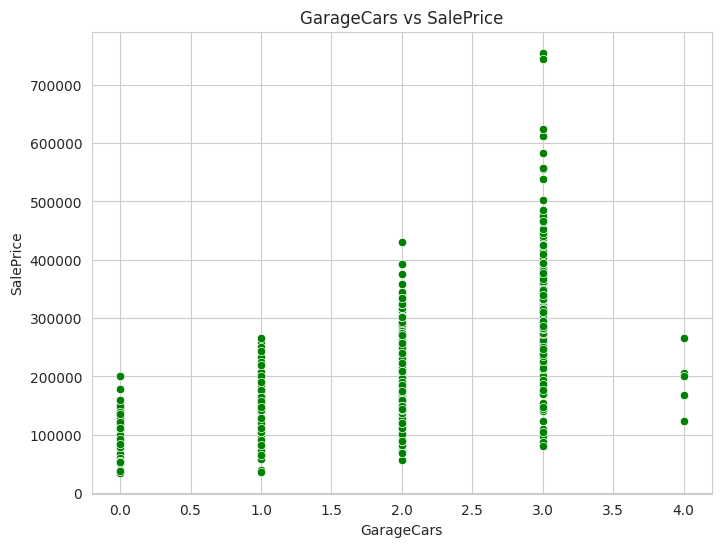
The distribution of the sales price was found to be largely populated around 150000, with a positively skewed distribution.



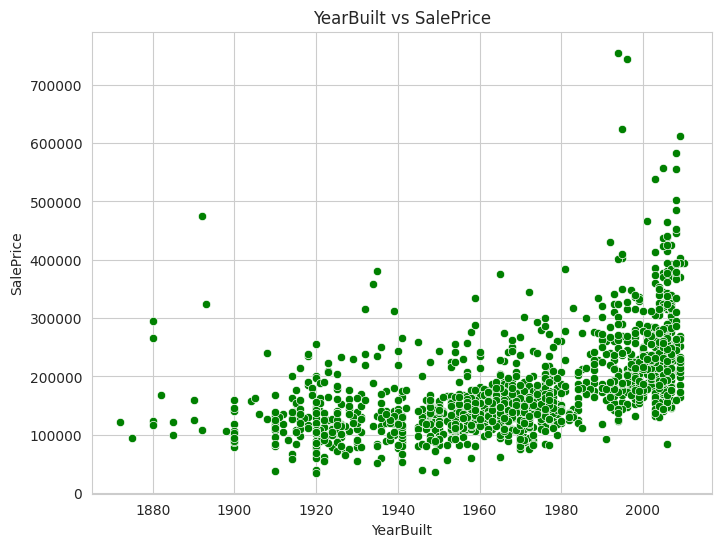
OverallQual seems very directly positively correlated to SalesPrice, and the graph shows that most of the entries for OverallQual range from 3-8.



Sales price increases with the number of the FullBath category. Most living areas are between 1000 and 2500 for GrLivArea. GrLiveArea fits a log transformation better, which shows a more prominent direct positive correlation, which is intuitive. A similar correlation also applies to TotalBsmtSF.



GarageCars does not contain many entries above 4, which also shows a break away from the trend of higher sales prices with an increase in GarageCars. There is a constant increase in sales price from GarageCars of 1 to 3. This could be impacted by the low number of samples for 4 GarageCars.



Sales price has a positive correlation with YearBuilt. This makes sense as newer houses should intuitively cost more than older houses. The newer a house is, there seems to be a more prominent rise in price.

### Creating New Features

**Examples of creating features**

df['TimeSinceRemodel'] = df['YrSold'] - df['YearRemodAdd'] # Calculate the difference between the year sold and year remodeled

df['TotalCarpetArea'] = df['BsmtFinSF1'] + df['BsmtFinSF2'] + df['1stFlrSF'] + df['2ndFlrSF']

df['TotalBathrooms'] = df['FullBath'] + df['BsmtFullBath'] + (0.5 \* df['HalfBath']) + (0.5 \* df['BsmtHalfBath'])

Age Calculation: New features are created to represent the age of the property, age of the garage, and time since last remodel. These features capture temporal information, which can be important predictors of house prices.

Total Area Calculation: Total carpet area, total porch area, total outdoor area, and total finished basement area are calculated by aggregating relevant features. These features provide comprehensive measures of the size and amenities of the properties.

Composite Features: Composite features such as overall quality and condition composite score, presence of amenities (pools, garages, fireplaces), and age of property at sale are created by combining multiple individual features. These composite features capture interactions and relationships among different aspects of the properties.

Total Number of Rooms: The total number of rooms is calculated by summing the number of bedrooms, full bathrooms, and half bathrooms. This feature provides insights into the overall size and functionality of the properties.

Years Since Last Remodel: This feature captures the duration since the last remodeling of the property, which can influence its market value.

Presence of Amenities: Binary indicators are created to denote the presence of pools, garages, and fireplaces. These features capture the availability of desirable amenities, which can impact house prices.

Overall Quality and Condition Composite Score: This feature is created by multiplying the overall quality and condition ratings. It represents an aggregated measure of the overall condition of the properties.

Total Number of Floors: The total number of floors is calculated by adding one to the number of rooms above ground level. This feature accounts for multi-story properties and their potential impact on prices.

## Gradient Boosting

In our code, we used Gradient Boosting as it presented the model with the lowest RMSE. We had also tried Random Forest as a viable alterative, but Gradient Boosting offered a greater RMSE value.

In this. we initialized a GradientBoostingRegressor instance with default or predefined hyperparameters. Then, the instance is initialized with hyperparameters obtained from the optimization process (Optuna).

Model Training:

The initialized GradientBoostingRegressor is trained on the training data (X\_train) along with the corresponding target variable (y\_train). It learns to predict the logarithm of the target variable SalePrice to handle its skewed distribution.

Gradient Boosting provides a measure of feature importance, indicating the contribution of each feature to the predictive performance. This is used to retain the most informative features for model training and prediction.

## Using Optuna for Optimising

An objective function (objective) is defined for evaluating the performance of the Gradient Boosting Regressor for a given set of hyperparameters. For this competition, the emphasis was on minimizing the root mean squared error (RMSE) value. Hence, the objective function computes the negative RMSE using cross-validation on the training data.Range or distribution constraints are specified for each hyperparameter to guide the optimization process.

Optimization Process:

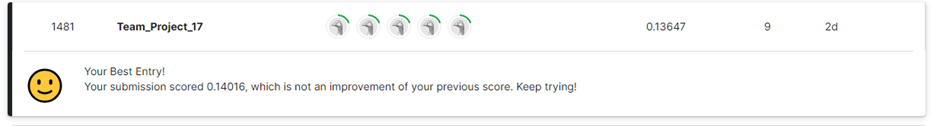
Optuna executes the optimization process, iteratively exploring different sets of hyperparameters based on the defined search space. We performed a search study with 10 trials, as it provided the best balance between performance and tuning performance.

Best hyperparameters: {'learning\_rate': 0.08158356209338379, 'n\_estimators': 105, 'max\_depth': 71, 'min\_samples\_split': 119, 'min\_samples\_leaf': 14}

Best score: -0.13239006948450863

These optimized hyperparameters are then used to initialize the final GradientBoostingRegressor model for prediction. By combining Gradient Boosting with Optuna, our code effectively achieves improved predictive accuracy by leveraging ensemble learning and automated hyperparameter optimization techniques for house price prediction.

### Leaderboard Position on Kaggle



### Results

**Output Predictions**

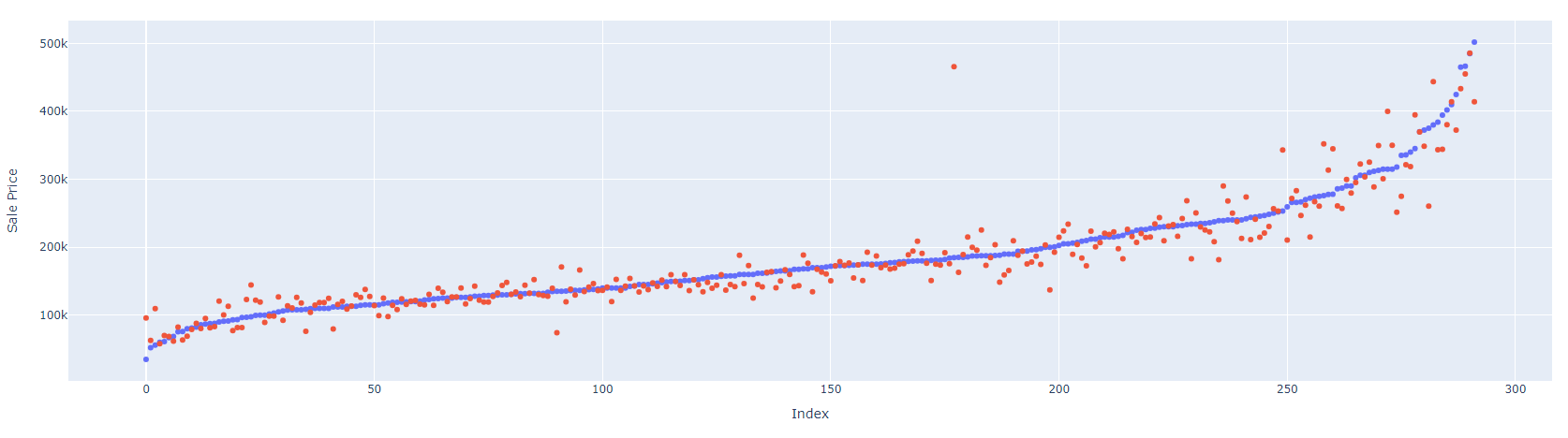
**The first 50 cells are produced here:**

| **pred\_xgb** |
| --- |
| **130164.4** |
| **165058.9** |
| **167258.1** |
| **191353.8** |
| **185800** |
| **178819.3** |
| **176150.6** |
| **173715.6** |
| **176422.6** |
| **115962.7** |
| **202498.6** |
| **96418.93** |
| **105827.8** |
| **147626.7** |
| **132335.7** |
| **385139.1** |
| **246833.3** |
| **292619.6** |
| **212803.7** |
| **460822.5** |
| **309497.2** |
| **203672.4** |
| **159094.7** |
| **163207.3** |
| **174047.6** |
| **203011.1** |
| **324165.5** |
| **243841.5** |
| **212977.8** |
| **215203.1** |
| **184090.2** |
| **94257.16** |
| **188989.6** |
| **291172.3** |
| **298238.6** |
| **215519.2** |
| **181726.4** |
| **152624.9** |
| **150509.3** |
| **144844.9** |
| **172217** |
| **154859.4** |
| **308162.9** |
| **236852.7** |
| **229993.3** |
| **161882.4** |
| **216654.9** |
| **195942.6** |
| **164841.2** |

The initial output of the predictions is as an array:  
  
array([127788.8596193 , 166838.08288041, 172868.90255025, ...,

179281.48839666, 104356.98825588, 207486.93628675])

**Prediction vs Actual**



The predicted values (blue) can be seen to be related to the trend shown by the red (actual) data points. It has fit itself to showcase a general inclination to the trend, irrespective of the variabilities in the data.

**RMSE:** 0.1484

## Member Contributions

Alireza Bavafa

Contributions:

In this project, I focused on the most important data preprocessing task, which is feature engineering during the data preprocessing phase. This procedure enhances the model's performance by inducing new informative features, which are derived from the old ones. I started with imputing missing values in the key columns, namely 'LotFrontage', 'MasVnrArea', and 'GarageYrBlt', so that the dataset is made complete and robust for modeling. One-hot encoding for the categorical variables was done to convert them to a numeric format, making them ready to be given as input to machine learning algorithms. Further, I carried out the derivation of new features, such as 'TotalSF', which sums up 'TotalBsmtSF', '1stFlrSF', and '2ndFlrSF', to achieve the sum of the total area of the property in this case, which will act as a surrogate for its size. Therefore, this process not only improved the quality of the dataset but also enriched the dataset with new dimensions, which are likely to have significant effects on the target variable, 'SalePrice'.

Learnings:

From my other classes, I have learned that the model is fancy, but it's the tuning parameters that give high performance, and the significance of feature selection and engineering is no less. Very careful selection and engineering of the features make a huge difference to the accuracy and interpretability of the model. This project has been very important in bringing into perspective how a raw dataset can be converted into meaningful predictors for the purpose of empowering a machine learning model. This project has reinvigorated the idea that a well-thought-out feature selection strategy is as pivotal as model and hyperparameter selection and tuning in driving predictive modelling to successful completion.

Akshay Sharma:

Contributions:

I developed a predictive model using Gradient Boosting, focusing on improving the model's accuracy using feature engineering and exploratory data analysis. I used cross-validation techniques to ensure the model worked well on different parts of the data, ensuring its reliability. I also prepared thorough documentation of all the steps from start to finish, which helped keep the project transparent and easy to follow.

Learnings:

I improved my machine learning skills, particularly in using advanced methods like Gradient Boosting and understanding the importance of setting model parameters correctly. I learned to visualize data effectively in order to identify patterns and relationships more easily. Another vital learning area was understanding the importance of thorough testing to validate the model's accuracy across different data samples. I explored how different features impact the model's predictions and practiced explaining these effects using tools designed for model interpretation.

Solayappan Ganesaan

Contributions:

I contributed to using a new framework designed for hyperparameter tuning, Optina, which greatly improved the general performance of the model. Optuna was used to optimise the study of the variables using an objective function. I also c

Learnings:

I acquired skills in training and optimising machine learning models using cloud-based resources, including hyperparameter optimization techniques like Optuna. Leveraging GPU/TPU instances for training accelerated model development and experimentation. Through collaboration using cloud-based tools like GitHub and shared notebooks, I enhanced my teamwork skills and adopted version control best practices. This facilitated seamless collaboration and efficient code management.

Aagney Menon

Contribution:

I originally started the programming of the code, starting off with using TFDF and Keras for housing price prediction. I also contributed to the cleaning of data, especially for the addition of combined features for improving model performance. I participated in conducting some of the EDA and encoding of variables for categorical variables as well. I also worked on creating a Ranom Forest model to test the model, and using some of the hypertuning measures to get an optimal result.

Learnings:

I learned to apply domain knowledge to understand the variables in real life data to understand the impact of variables on the target variable, if it is intuitive as such, and what variables could be combined in order to improve a model. I also learned how to compare models under different criteria and how to hypertune TFDF models in the attempt to learn the code.

Harish Bhupathiraju

Contribution:  
  
I worked on exploratory data analysis and encoding and imputing the data. I used the encoding to address missing values in categorical variables by assigning a specific value (e.g., "NA") to represent missing data and ensure that both the categorical and numerical variables become complete and suitable for model training without the need for imputation. I saw to it that this encoding ensures consistency in data preprocessing across training and test datasets, as the same encoding scheme is applied to both. I also worked on the Random Forest model hypertuning which was originally tested to check for optimal outputs.

Learning:

Through my involvement in this project, I've learned how to explore datasets thoroughly, identifying patterns and potential issues, and ensuring data quality. Encoding categorical variables and handling missing values have taught me essential data preprocessing techniques, ensuring consistency across training and test datasets. Hyperparameter tuning has shown me the importance of optimizing model parameters for better performance, while model evaluation has helped me assess and compare different models effectively, essential for addressing real-world data science challenges.