Project Title

House Price Prediction using Machine Learning Techniques

(Due Date: 8th December 2023)

Course Name: Machine Learning

Instructor Name: Salim Sazzed

Project By

Pranay Kaira	U00886947
Sruthi Gosukula	U00865831
Vishnu Choundur	U00869233
Vishnu Teja Yerrapragada	U00869711

Table of Contents

Abstract

This report presents a comprehensive analysis of housing price prediction using a detailed Kaggle dataset. The project applies machine learning techniques to identify key factors that influence housing prices. The objective is to create a predictive model that accurately estimates home values based on a variety of features, such as location, size, and amenities. Our approach synthesizes vast amounts of data to assist stakeholders in making informed decisions in the property market. The findings contribute to a deeper understanding of market trends and offer a predictive outlook for potential investors and homebuyers.

Introduction

In this report, we explore the task of predicting house sale prices using a dataset from Kaggle, which we thoroughly prepared for analysis. Our approach involved cleaning the data, strategically selecting relevant features, and using visualizations to gain insights. We experimented with various prediction models, from standard statistical techniques to more complex deep learning algorithms, fine-tuning them for optimal accuracy. The aim was to develop a reliable model for estimating house sale prices, offering practical insights for understanding and navigating the housing market.

Problem Statement:
The goal of this project is to develop a robust model that can accurately predict house prices based on relevant features. The model should be capable of generalizing well to unseen data and provide valuable insights into the factors influencing house prices.

Literature Survey

We have extensively researched various literature, online resources, and academic papers to guide the approach and methodology for our project on housing price prediction. Specifically, we have referred to key sources such as:

The House Price Index from the Federal Housing Finance Agency, which provided valuable macro-level insights into housing market trends and price fluctuations over time. The study by Fan, Cui, and Zhong on 'House Prices Prediction with Machine Learning Algorithms,' published in the ICMLC 2018 proceedings, which offered a comprehensive understanding of different machine learning techniques applied in this domain.

The research by Phan TD on 'Housing Price Prediction Using Machine Learning Algorithms: The Case of Melbourne City, Australia,' from the 2018 International Conference on Machine Learning and Data Engineering, which presented a case-specific application of these methodologies in a distinct geographical context.

The paper by Mu, Wu, and Zhang on 'Housing Value Forecasting Based on Machine Learning Methods,' which provided insights into the application of various predictive models in housing valuation. These resources have been instrumental in shaping my understanding of the current state of machine learning applications in real estate pricing. They have guided the selection of algorithms, data preprocessing techniques, and model validation methods used in the project, ensuring that our approach is aligned with current best practices and innovations in the field."

Methodology

During the development and implementation of our project, we closely followed the machine learning life cycle. To ensure a thorough and organized process, our method comprised going step-by-step through every stage of the life cycle. From the formulation of the problem and the gathering of data to the training, assessment, and implementation of models.

1. Problem Definition
2. Data Collection
3. Exploratory Data Analysis (EDA)
4. Data Preprocessing
5. Transformation
6. Feature Engineering
7. Model Selection
8. Model Training
9. Model Evaluation
10. Model Deployment

Data Collection

The dataset utilized is sourced from Kaggle.com, and the dataset link is provided for reference.

Dataset Link: https://www.kaggle.com/c/house-prices-advanced-regressiontechniques

Data Preprocessing

In the context of our project, the dataset displays a hierarchical structure with a total of 1460 rows and 81 columns, each of which represents unique variables related to residential properties. 43 of the variables in this large set are categorical in nature, representing the qualitative components of the dataset, while the remaining 38 variables are numerical in nature, representing the quantitative aspects.

A thorough examination of the many factors affecting property prices is made possible by this organized framework. While the numerical variables include quantitative indicators like lot characteristics and building kinds, the categorical variables provide insights into qualitative aspects like zoning classifications and neighborhood details. A more in-depth examination is made possible by the clear classification, which also makes it possible to create a forecast model that accurately reflects the intricacy of the housing market.

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	Mi
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	Shed	
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	

1460 rows × 81 columns

Data Cleaning:

A precise data cleaning procedure was carried out in the first stage of our project to ensure the accuracy and consistency of the dataset. The steps following were carried out in a sequential manner.

1. Handling Missing Values
2. Outlier Detection and Treatment
3. Dealing with Duplicates
4. Handling Categorical Data
5. Addressing Data Inconsistencies
6. Normalization/Scaling

1. Handling Missing Values:

In order to conduct a thorough analysis, missing data must be addressed. The identification of missing values was done with great care, and suitable techniques, like imputation or elimination, were used to lessen their impact on the results of later studies.

Handling missing values of Numerical Data:

Columns with missing values in the numerical data include GarageYrBlt, LotFrontage, and MasVnrArea. We used two imputation methods to fill in these gaps.

Numerical variables

```
[7]: nume_cols = housing.select_dtypes(include=['number'])
     print("Number of Numerical Columns   : ",len(nume_cols.columns))
```

```
Number of Numerical Columns   : 37
```

```
[8]: nume_cols.describe()
```

[8]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch
count	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	1460.000000	...	1460.000000	1460.000000	1460.000000	1460.000000
mean	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	46.549315	...	94.244521	46.660274	21.954110	3.409589
std	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	161.319273	...	125.338794	66.256028	61.119149	29.317331
min	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
25%	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000
50%	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0.000000	...	0.000000	25.000000	0.000000	0.000000
75%	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	0.000000	...	168.000000	68.000000	0.000000	0.000000
max	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000	...	857.000000	547.000000	552.000000	508.000000

8 rows × 37 columns

Iterative Imputation: The missing values in the targeted columns were estimated and replaced using an advanced iterative imputation technique. Several rounds of imputation are used in this iterative process to gradually improve the predictions and fill in the missing data with increasingly precise information.

Considering the unique nature of the "GarageYrBlt" variable, it may not be best to use the Iterative Imputation approach to address missing values in this variable. "GarageYrBlt" is the year a garage was constructed; it is a discrete number variable that denotes particular years. When handling missing years in the context of time-based data, iterative imputation which is frequently employed for continuous or categorical variables might not be the best option.

Employing the iterative approach to impute missing years in "GarageYrBlt" may lead to imputed values that are not contextually relevant and may not follow the data's chronological order. By comparison, using regression imputations is a better strategy for "GarageYrBlt". Regression imputation is a more appropriate method for imputing missing values in a variable such as "GarageYrBlt," which is naturally associated with certain years and a chronological sequence, because it takes into account the relationships between the variables.

Regression Imputation:

In order to infer and replace missing values, regression-based imputation was also applied. By predicting and imputing the missing values using the relationships between the variables, this technique improves the dataset's precision.

To handle the missing variables in "GarageYrBlt," we employ the regression imputation procedure in a methodical manner. Using other relevant variables as independent predictors, we aim to construct a regression model with "GarageYrBlt" as the dependent variable. Using data from rows where "GarageYrBlt" values are available, the model is carefully trained. The values of the predictor variables in rows where "GarageYrBlt" is missing are then used to accurately forecast missing values of "GarageYrBlt" using the trained model.

Strong correlations between variables are taken into account by this technique, which leads to more accurate imputations. This is especially true in cases when "GarageYrBlt" and the selected predictors have a strong correlation. The correlation between "GarageYrBlt" and all other numerical variables in the dataset is one of the key steps in doing correlation analysis, which is necessary to validate this link.

Upon a thorough analysis of the summary, we have determined that YearBuilt, YearRemodAdd, GarageCars, GarageArea, OverallQual, and OverallCond are important variables that have a significant link with "GarageYrBlt." These variables' correlations highlight how important they are as reliable predictors in the regression model. The reliability and efficacy of our regression imputation technique for "GarageYrBlt" are improved by this thorough analysis.

Evaluating the Relationship between "GarageYrBlt" and Predictors:

To assess the strength of the relationship between "GarageYrBlt" and potential predictors, we conducted a comprehensive correlation analysis. This required figuring out how "GarageYrBlt" and every other numerical variable in the dataset were correlated. Those variables that show a strong positive or negative connection with "GarageYrBlt" are considered as possible predictor candidates. Strong predictors for our research were chosen based on the strength and direction of these correlations, which offer insightful information about the degree of relationship between "GarageYrBlt" and particular factors.

Handling missing values of Categorical variables:

Categorical variable

```
cat_cols = housing.select_dtypes(exclude=['number'])
print("Number of Categorical Columns : ",len(cat_cols.columns))
```

```
Number of Categorical Columns :  43
```

```
cat_cols.describe()
```

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	...	GarageType	GarageFinish	Garag
count	1460	1460	91	1460	1460	1460	1460	1460	1460	1460	...	1379	1379	
unique	5	2	2	4	4	2	5	3	25	9	...	6	3	
top	RL	Pave	Grvl	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Norm	...	Attchd	Unf	
freq	1151	1454	50	925	1311	1459	1052	1382	225	1260	...	870	605	

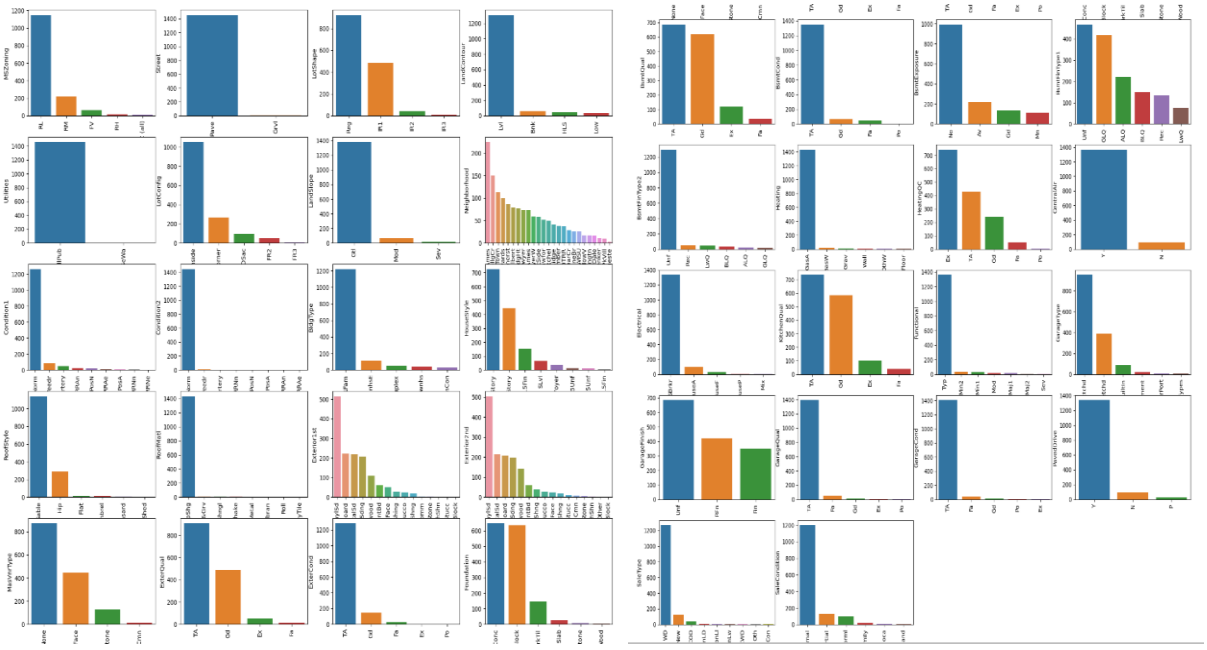
4 rows × 43 columns

We can employ Mode Imputation to deal with the missing values in these variables.

Mode Imputation:

This involves replacing the mode which stands for the most prevalent category within the corresponding variable with any missing values. One popular technique for dealing with missing categorical data is mode imputation.In this case, certain factors influence the mode imputation selection. The missing categorical variables in our dataset do not appear to be related to any other predictors. These variables' missingness is completely arbitrary and unconnected to the values of other variables. In light of this situation, mode imputation appears to be a sensible and practical method for dealing with the missing values in these categorical variables.

Graphical Representation of Categorical Data:



In order to understand the distribution and patterns within the data, categorical variables must be shown. This is an overview of analyzing categorical variables.

Outlier Detection and Treatment:

To improve the resiliency of our dataset, we utilized the Z distribution technique to detect and eliminate anomalies. We were able to systematically target data points that deviated significantly from the mean by setting a cut-off alpha value of 0.01—which is equal to a Z-score of 3. This strategy makes sense statistically since data points that are outside of the 99th (or 1st) percentile of the Z distribution are referred to as outliers. We wanted to make sure that extreme numbers that would distort our analysis and model training were eliminated, so we put this strict requirement into practice.This method not only contributes to the overall integrity of the dataset but also aligns with the principle of mitigating the influence of outliers on subsequent statistical analyses and machine learning models.

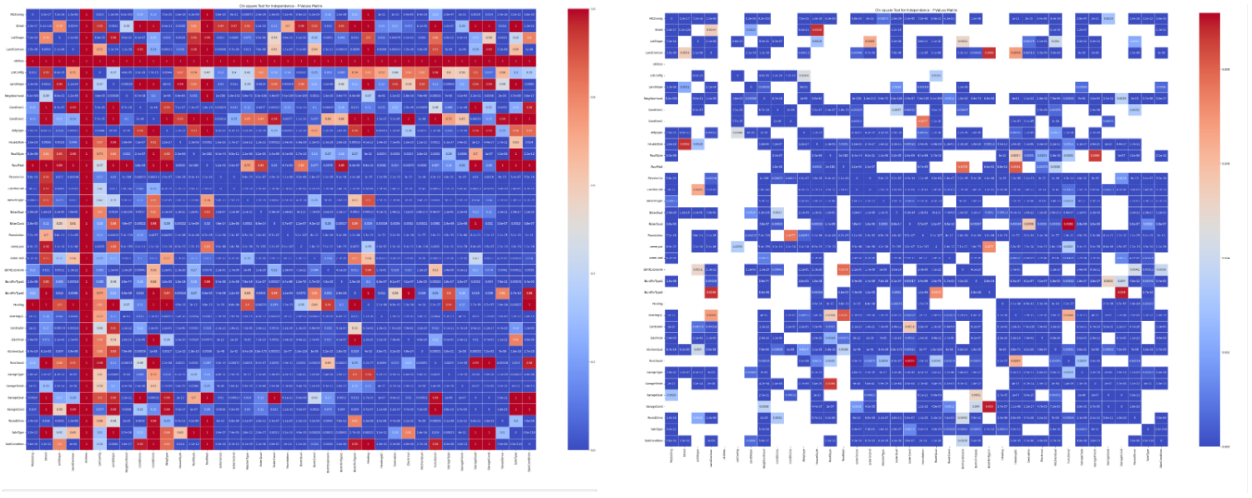
[34]:
```
import pandas as pd
from scipy import stats
import numpy as np

# Assuming your DataFrame is named 'df'
# Replace 'df' with the actual name of your DataFrame

# Function to remove outliers using Z-score for all columns
def remove_outliers_zscore(df, threshold=3):
    df1 = df
    numerical_columns = df.select_dtypes(include=['number']).columns
    z_scores = np.abs(stats.zscore(df[numerical_columns]))
    outlier_rows = (z_scores > threshold).any(axis=1)
    df_no_outliers = df1[~outlier_rows]
    # Keep all columns (numerical and categorical) in the final output
    #df_no_outliers = pd.concat([df_no_outliers_numeric, df.select_dtypes(exclude=['number'])], axis=
    return df_no_outliers

# Apply the function to your DataFrame
df_no_outliers = remove_outliers_zscore(fn_df1)

# Display the shape before and after removing outliers
print("Original DataFrame shape:", fn_df1.shape)
print("DataFrame shape after removing outliers:", df_no_outliers.shape)
```

```
Original DataFrame shape: (1460, 75)
DataFrame shape after removing outliers: (1015, 75)
```

[35]:
```
df1 = df_no_outliers
df1
```

Chi-Square Test for Categorical Variables:

To look into the association between the categorical variables in our dataset, we used the Chi-Square test. This test is especially useful for evaluating the degree of variable independence and detecting significant differences between observed and predicted frequencies.



Plotting the Chi-Square test results revealed multicollinearity, or the existence of patterns and correlations between several categorical variables. Given that multicollinearity may have an effect on the dependability of Chi-Square results, a deliberate choice was made to exclude variables that showed signs of this phenomena. In order to strengthen the analysis's robustness and guarantee that the findings appropriately reflect the real relationships between the remaining categorical variables, it is essential to identify and exclude variables that exhibit multicollinearity. A more accurate interpretation of the relationships within the dataset is fostered by this methodical approach, which also improves the integrity of the Chi-Square test results.

The right image shows the association between the two variables. On the left image, we have invisibled the variables that are not associated (p value > 0.01). We have visible variables that are more associated with other variables (p value < 0.01).

In the initial plot, we also observed variables displaying multicollinearity. Subsequently, in the second plot, we removed those specific variables to address the issue.

ANOVA Testing:

We used ANOVA (Analysis of Variance) testing to investigate the association between our dataset's target variable and a categorical variable. Using this method, one can determine whether there are significant differences between the means of the target variable at various levels of the category variable.

TRANSFORMATION

In this phase, we will convert the categorical variables transformed using Label encoding. For deep learning models i.e., CNN, We have scaled the numerical variables using standardScaler method.

Encoding:

Converting the cleaned category columns into numerical representations is necessary for ensuring consistency throughout the dataset. Label Encoding is the technique we have selected for this conversion.

Label Encoding:

In order to ensure consistency across the entire dataset and facilitate the implementation of machine learning models, it is imperative to convert cleaned categorical columns into numerical format. This conversion is vital because machine learning algorithms typically require all variables to be in numerical form.

Label encoding has been selected for this purpose, primarily due to its simplicity and ability to reduce dimensionality. Unlike one-hot encoding, which generates binary variables for each category and substantially increases dimensionality, label encoding assigns numerical labels to categorical variables. This results in a more streamlined representation of the data, contributing to computational efficiency.

While one-hot encoding is a potent technique for handling categorical variables, it was not employed in this analysis for several reasons. One-hot encoding introduces a multitude of binary variables, leading to a significant expansion in dimensionality. Moreover, it can give rise to multicollinearity issues, as the presence of one binary variable implies the absence of others.

To maintain dataset efficiency, reduce dimensionality, and preserve ordinal information where applicable, label encoding emerged as a more suitable alternative for our modeling purposes. Label encoding strikes a balance between efficiency and dimensionality, aligning well with our analytical objectives.

Notably, label encoding retains the order in categorical variables, making it particularly beneficial for those with ordinal characteristics. This characteristic aligns seamlessly with our modeling objectives, contributing to the suitability of label encoding for our analytical needs. In summary, the decision to choose label encoding is grounded in its efficiency, dimensionality reduction, and compatibility with the ordinal nature of certain categorical variables in our dataset.

Feature Selection

The process of feature selection, which involves identifying and keeping the most pertinent variables, is essential to creating predictive models that work. It seeks to decrease complexity, increase interpretability, and boost model performance. The overall forecast accuracy is greatly increased by applying a variety of strategies to choose characteristics according to their significance. The analysis's objectives and the type of data it contains determine which feature selection techniques are used.

Integration of Data Frames and Feature Selection:

The following procedure of our research included unifying the two data frames and combining their contents into a single, comprehensive dataset. This combination offered a cohesive basis for our modeling efforts, utilizing the advantages of both numerical and categorical data.

```
combined_df = pd.concat([nume_cols,cat_cols],axis=1)
```

- The concat function is used to combine the dataframes.
- For Numerical feature selection we used lasso regression and for Categorical feature selection we employed Random forest regression.

Lasso Regression:

We used the Lasso (Least Absolute Shrinkage and Selection Operator) approach to choose numerical characteristics. As a regularization technique, Lasso efficiently finds and keeps the most significant numerical features while punishing the less significant ones. This feature selection method improves our models' effectiveness and interpretability.

Advantage of Using Lasso Regression for Numerical Variable:

- L1 regularization
- Feature Importance
- Dealing with multicollinearity
- Simplicity and interpretability

```
# Print results
print("Lasso Regression:")
lasso_feature = X.columns[lasso_model.coef_ != 0]
print("Selected Features:", lasso_feature)
print("Best alpha:", lasso_model.alpha_)
print("R-squared (R2):", lasso_r2)
```

```
Lasso Regression:
Selected Features: Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'TotalBsmtSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'HalfBath', 'BedroomAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt',
       'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', 'ScreenPorch', 'MiscVal', 'MoSold', 'YrSold'],
      dtype='object')
Best alpha: 65.79332246575683
R-squared (R2): 0.8784414283865694
```

Random Forest Regressor:

We opted to employ the Random Forest Regressor for selection when it came to categorical features. This approach makes use of the strength of group learning to determine the significance of categorical variables. The Random Forest Regressor helps choose the most relevant category characteristics by evaluating each variable's contribution to prediction accuracy.

Our combined dataset included the features that were chosen from the numerical and categorical domains. Applying predictive algorithms to this annotated information should produce more reliable and knowledgeable results, setting the stage for a thorough investigation.

Advantage of Using Random Forest Regression for Numerical Variable:

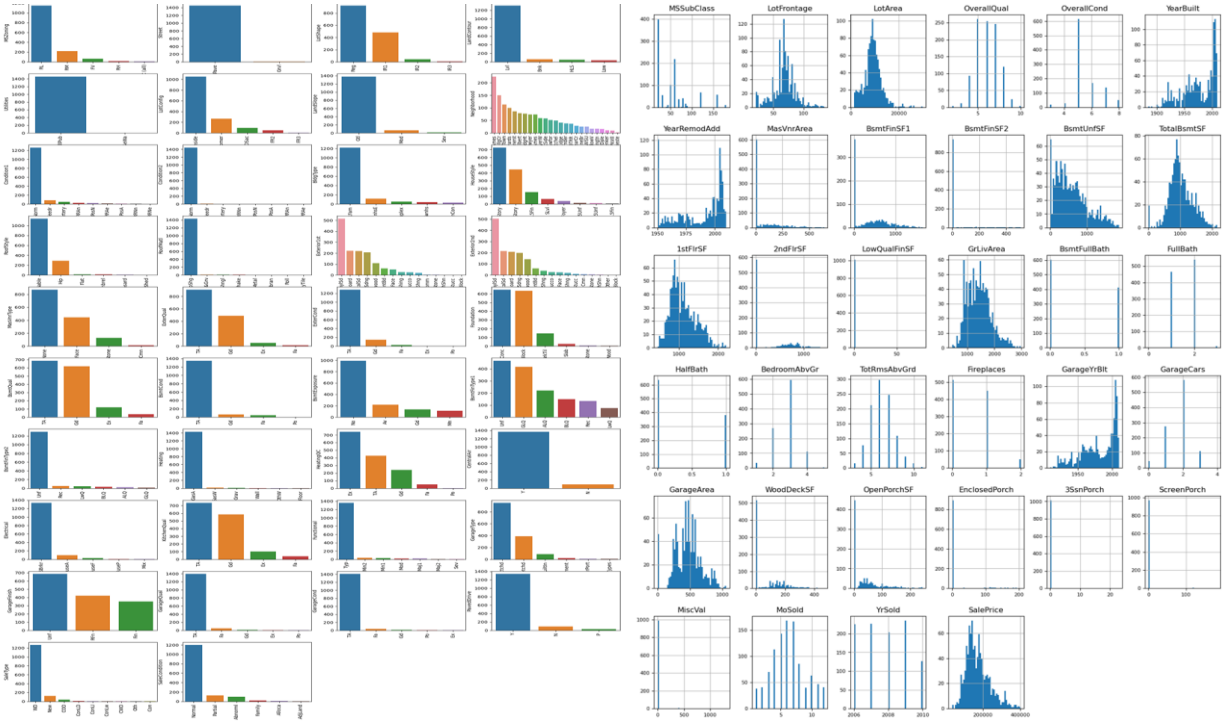
- Non Linearity and Interaction
- Feature Importance
- Robust to overfitting
- Ensemble Averaging

```
feature_importance_df['Feature']
```

```
']: 0           MSZoning
    1             Street
    2           LotShape
    3        LandContour
    4          Utilities
    5          LotConfig
    6          LandSlope
    7       Neighborhood
    8         Condition1
    9         Condition2
    10          BldgType
    11        HouseStyle
    12         RoofStyle
    13           RoofMatl
    14       Exterior1st
    15       Exterior2nd
    16        MasVnrType
    17          ExterQual
    18          ExterCond
    19         Foundation
    20          BsmtQual
    21          BsmtCond
    22      BsmtExposure
    23      BsmtFinType1
    24      BsmtFinType2
    25           Heating
    26         HeatingQC
    27        CentralAir
    28        Electrical
    29       KitchenQual
    30        Functional
    31        GarageType
    32      GarageFinish
    33        GarageQual
    34        GarageCond
    35        PavedDrive
    36          SaleType
    37     SaleCondition
    Name: Feature, dtype: object
```

Data Visualization

Here, we present the data visualization plots for both categorical and numerical variables.



Histogram for Categorical (Left) and Numerical Variable (Right)

Validation Techniques

Train-Test Split (80:20):

This approach entails dividing the dataset into two portions: an 80% training set for model training and a 20% test set for assessing the model's performance on previously unseen data. This foundational technique serves as a crucial step in evaluating the model's generalization capabilities.

Spliting the Data

```
[79]: from sklearn.model_selection import train_test_split
      import numpy as np
      # Separate the target variable and the features
      X = combined_df[f_feature].drop(columns=['SalePrice'])
      y = combined_df['SalePrice']

[80]: # Split the data into training and testing sets
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[81]: n = X_test.shape[0]
      p = X_test.shape[1]
      n,p

[81]: (203, 47)
```

K-Fold Cross-Validation:

In this model evaluation technique, the dataset is partitioned into K subsets, commonly known as folds. The model is trained and tested K times, with each iteration employing a different fold as the test set. This iterative process ensures a robust assessment of model performance across diverse subsets of the data.

5-Fold Cross-Validation:

A specific instance of K-Fold Cross-Validation with K set to 5. The dataset is divided into five subsets, and the model undergoes training and evaluation five times, utilizing a different subset as the test set in each iteration. This method offers a comprehensive evaluation, enhancing our understanding of the model's performance and generalizability across distinct partitions of the dataset.

These validation techniques collectively contribute to a comprehensive evaluation of the model, providing insights into its ability to generalize to new and unseen data.

Hyperparameters:

We cannot implement the Hyperparameters tuning for the Linear Regression. Can implement the Hyperparameters tuning for Random Forest, Gradient boosting Regression, Extreme Gradient (XG) Boosting Regression, and Cat Boosting. Based on the model and parameter used in the model, we used GridSearchCV with 5 fold validation to determine the best parameters that are suitable for the model. In order to determine the best parameters, we used mean square error as a scoring parameter.

```
param_grid = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}
```

```
# Create and fit a Random Forest regression model
rf_model = RandomForestRegressor(random_state=42)
```

```
# Perform Grid Search with cross-validation
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

# Fit the model to find the best hyperparameters
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and the corresponding accuracy score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print("Best Hyperparameters:", best_params)
print("Best Accuracy Score:", best_score)

# Evaluate the model on the test data with the best hyperparameters
best_rf_classifier = grid_search.best_estimator_
test_accuracy = best_rf_classifier.score(X_test, y_test)
print("Test Accuracy with Best Hyperparameters:", test_accuracy)
```

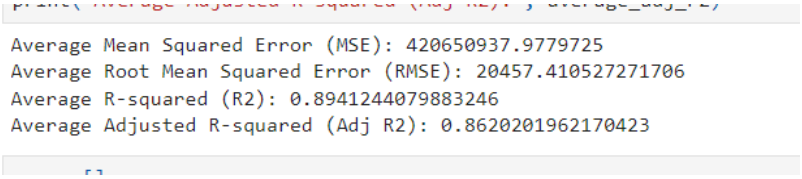
```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
Best Hyperparameters: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimato
rs': 100}
Best Accuracy Score: 0.8948028909301208
Test Accuracy with Best Hyperparameters: 0.8582799439618713
```

Models Implemented

1. Linear Regression
2. Random Forest Regression
3. Gradient Boosting Regression
4. XGBoost Regression
5. CatBoost Regression
6. Artificial Neural Network (ANN)
7. Feed Forward Neural Network (FNN)
8. Convolutional Neural Network (CNN)

Linear Regression:

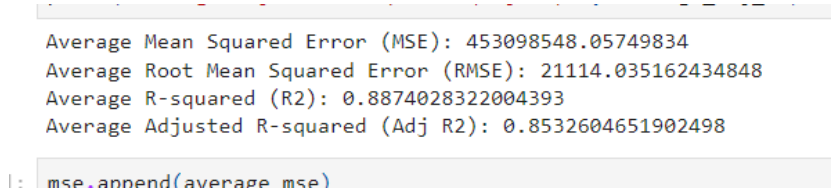
We've successfully applied linear regression and obtained the corresponding results for the following metrics.



```
Average Mean Squared Error (MSE): 420650937.9779725
Average Root Mean Squared Error (RMSE): 20457.410527271706
Average R-squared (R2): 0.8941244079883246
Average Adjusted R-squared (Adj R2): 0.8620201962170423
```

Random Forest Regression:

We've successfully applied Random Forest Regression and obtained the corresponding results for the following metrics.

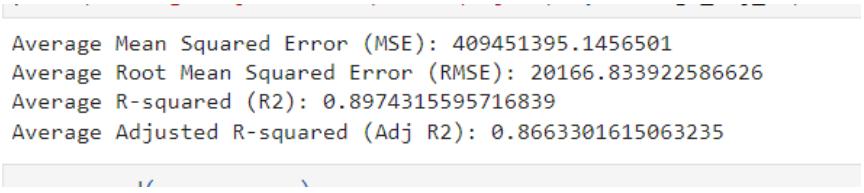


```
Average Mean Squared Error (MSE): 453098548.05749834
Average Root Mean Squared Error (RMSE): 21114.035162434848
Average R-squared (R2): 0.8874028322004393
Average Adjusted R-squared (Adj R2): 0.8532604651902498
```

```
mse.append(average_mse)
```

Gradient Boosting Regression:

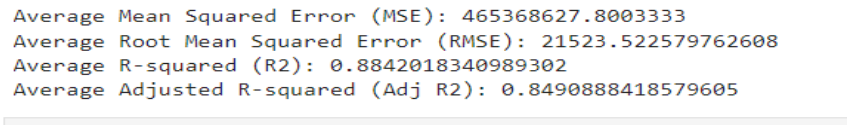
We've effectively implemented Gradient Boosting Regression and acquired the associated outcomes for the following metric.



```
Average Mean Squared Error (MSE): 409451395.1456501
Average Root Mean Squared Error (RMSE): 20166.833922586626
Average R-squared (R2): 0.8974315595716839
Average Adjusted R-squared (Adj R2): 0.8663301615063235
```

XGBoost Regression:

We've effectively implemented XGBoost Regression and acquired the associated outcomes for the following metric.



```
Average Mean Squared Error (MSE): 465368627.8003333
Average Root Mean Squared Error (RMSE): 21523.522579762608
Average R-squared (R2): 0.8842018340989302
Average Adjusted R-squared (Adj R2): 0.8490888418579605
```

CatBoost Regression:

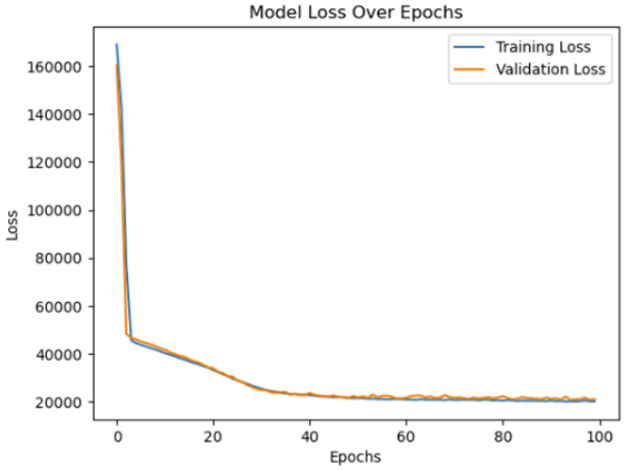
We've effectively implemented CatBoost Regression and acquired the associated outcomes for the following metric.

```
Average Mean Squared Error (MSE): 396216248.9777834
Average Root Mean Squared Error (RMSE): 19802.627727504434
Average R-squared (R2): 0.9017290696216117
Average Adjusted R-squared (Adj R2): 0.8719307875068745
```

Deep Learning:

Artificial Neural Network (ANN):

A computational model inspired by the structure and functioning of the human brain, consisting of interconnected nodes (neurons) organized in layers and used for various machine learning tasks, including regression.

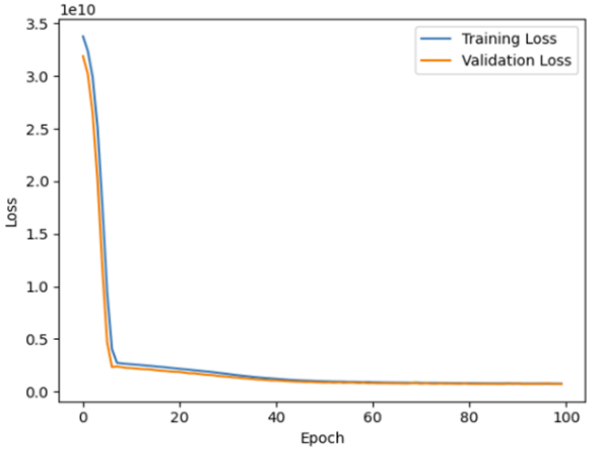



```
model1.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_23 (Dense)            (None, 50)                2400

 dense_24 (Dense)            (None, 25)                1275

 dense_25 (Dense)            (None, 50)                1300

 dense_26 (Dense)            (None, 1)                 51

=================================================================
Total params: 5026 (19.63 KB)
Trainable params: 5026 (19.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Feed Forward Neural Network (FNN):

A basic artificial neural network architecture where information moves in one direction, from input to output layer, without cycles or loops.

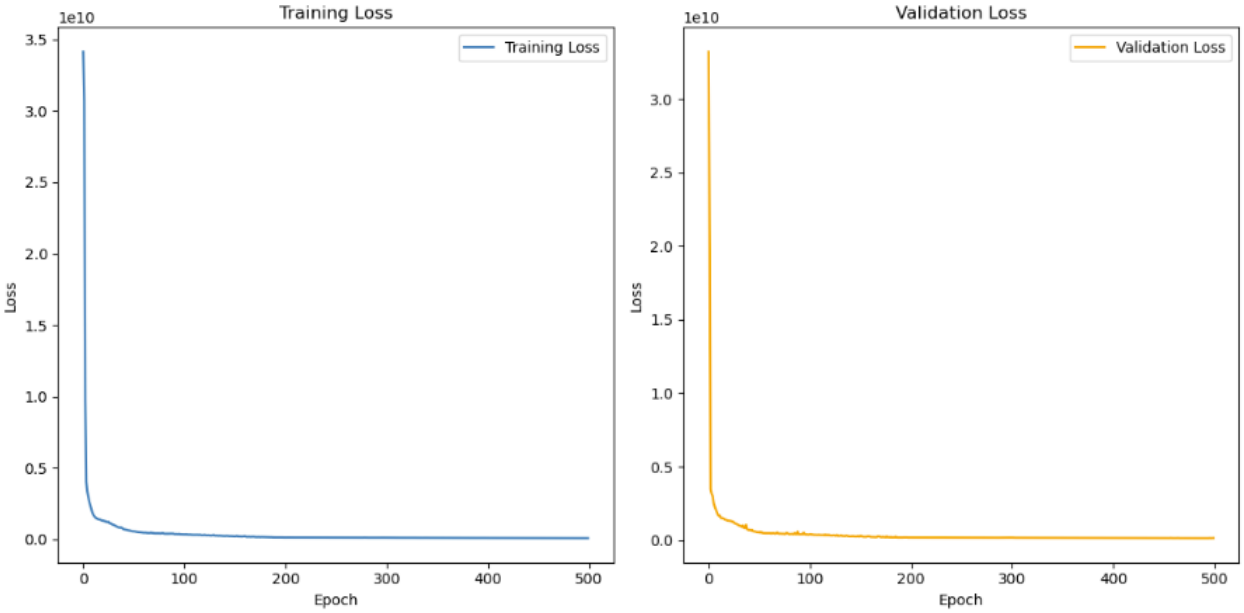



```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 64)                3072

 dense_11 (Dense)            (None, 32)                2080

 dense_12 (Dense)            (None, 1)                 33

=================================================================
Total params: 5185 (20.25 KB)
Trainable params: 5185 (20.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Convolutional Neural Network (CNN):

A type of artificial neural network, typically applied to analyze visual data, that uses convolutional layers to automatically and adaptively learn hierarchical representations.



Model: "sequential_1"

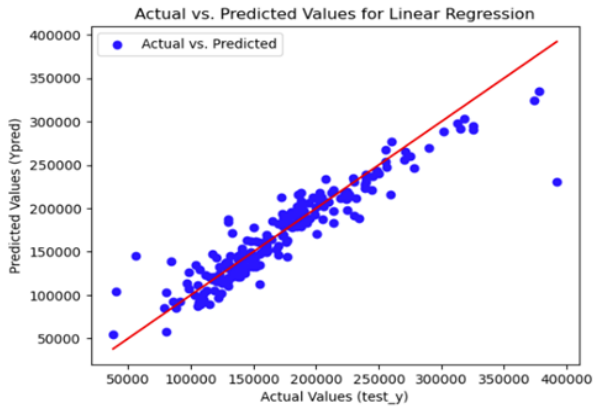
Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 147, 64)	256
max_pooling1d (MaxPooling1D)	(None, 73, 64)	0
conv1d_1 (Conv1D)	(None, 71, 128)	24704
max_pooling1d_1 (MaxPooling1D)	(None, 35, 128)	0
conv1d_2 (Conv1D)	(None, 33, 256)	98560
max_pooling1d_2 (MaxPooling1D)	(None, 16, 256)	0
flatten (Flatten)	(None, 4096)	0
dense_3 (Dense)	(None, 128)	524416
dense_4 (Dense)	(None, 64)	8256
dense_5 (Dense)	(None, 1)	65

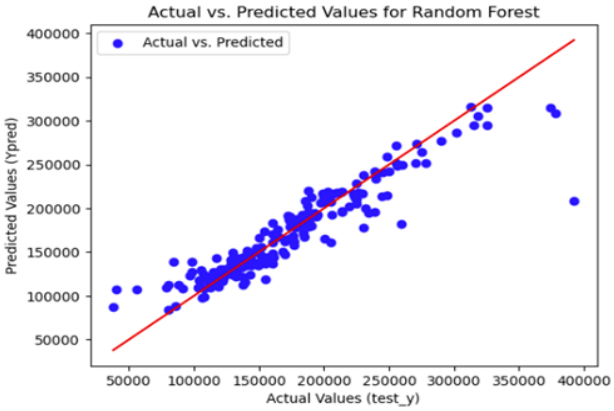
Total params: 656257 (2.50 MB)
Trainable params: 656257 (2.50 MB)
Non-trainable params: 0 (0.00 Byte)

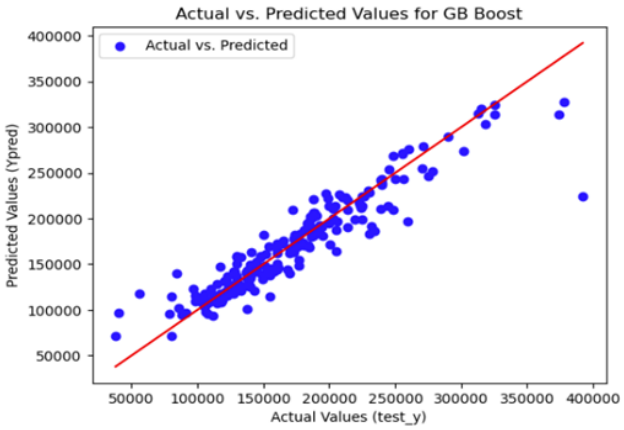
Comparison of the all the models implemented:

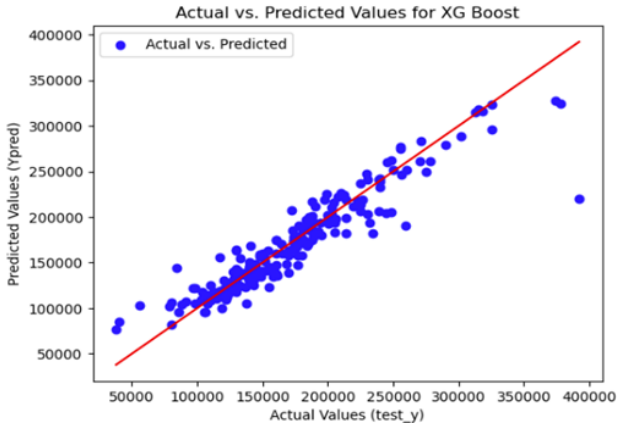
The implemented models' results reveal notable insights. Upon comparison, it is evident that the CNN model emerges as the frontrunner, showcasing a remarkable R^2 value of 96.56% and demonstrating exceptional predictive accuracy. Close contenders include CatBoost and XGBoost, with robust R^2 values of 90.56% and 90.07%, respectively. These findings emphasize the considerable performance and accuracy achieved by these models in effectively capturing and predicting patterns within the analyzed dataset.

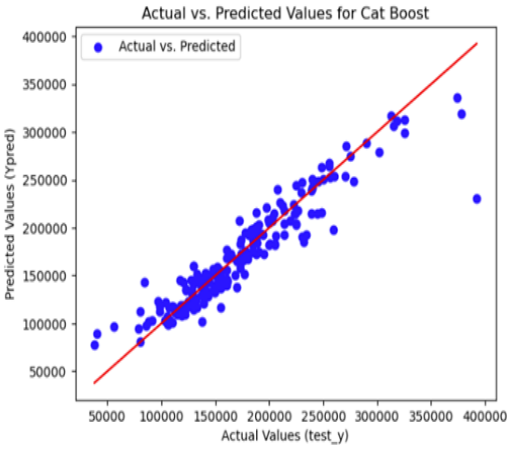
Machine Learning



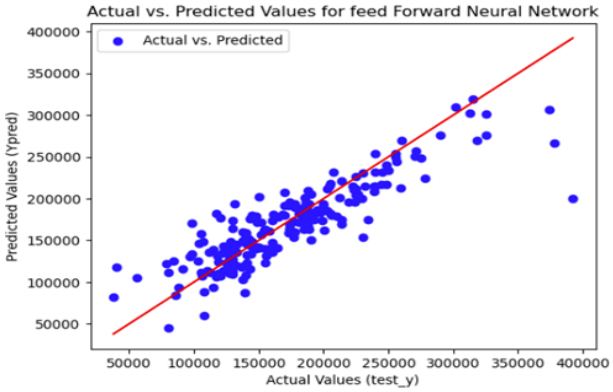


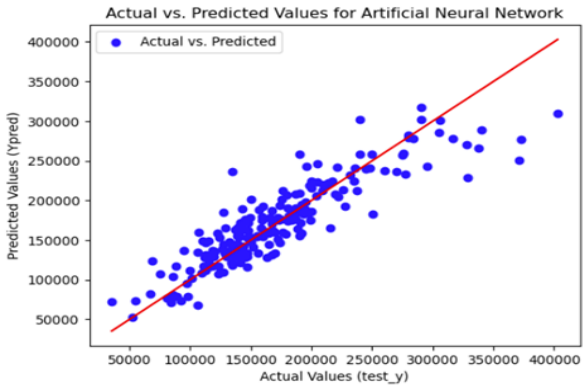


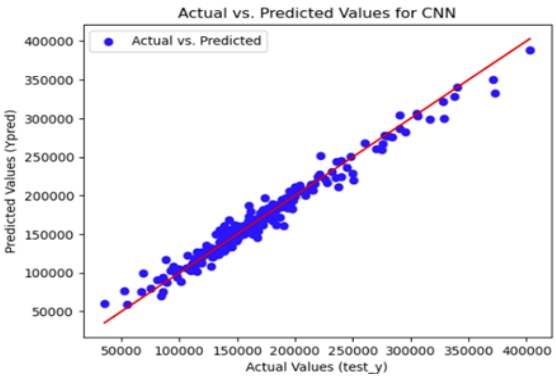




Deep Learning

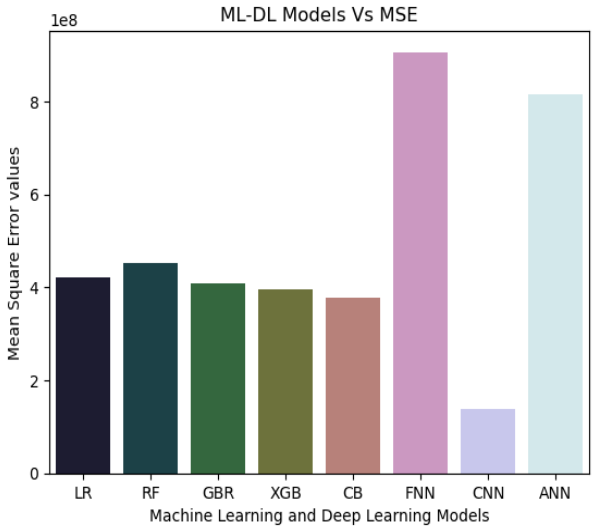


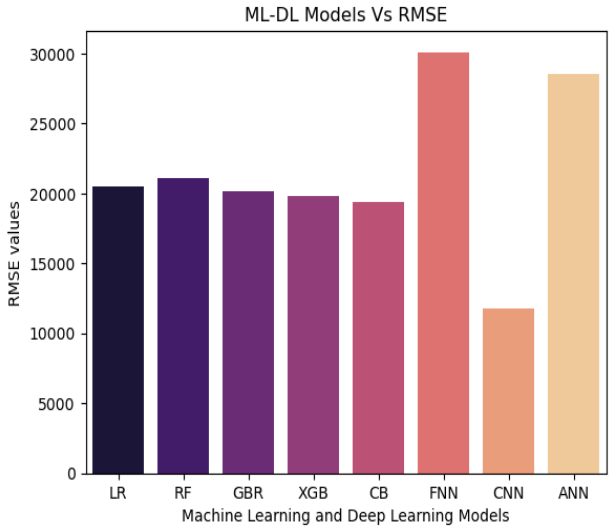


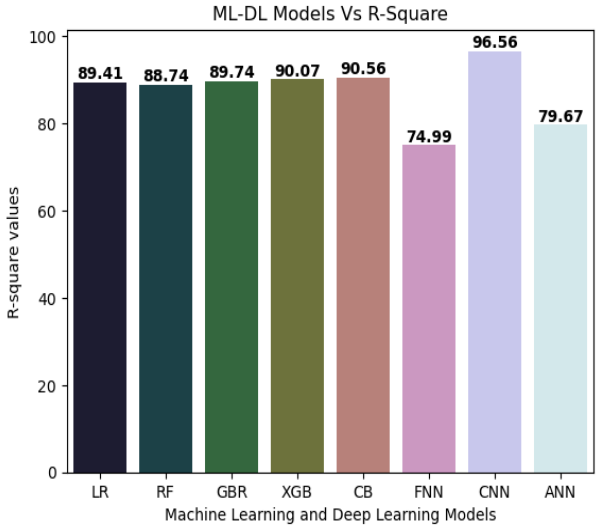


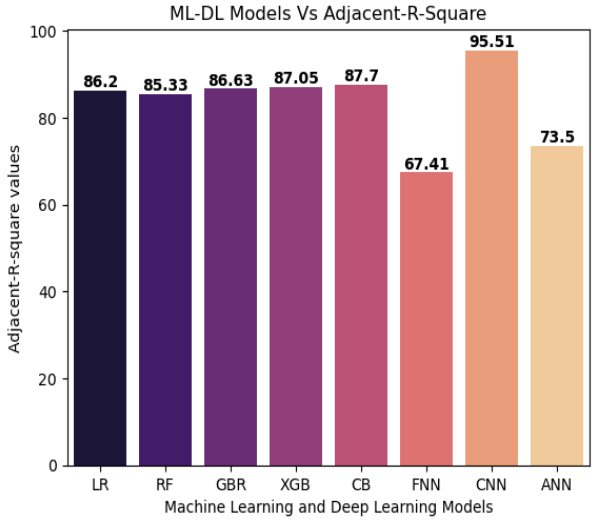
From above graphs we can see that the variance is low for the CNN when compared to other techniques. Next to CNN, we can see the Cat Boost and XGBoost models are working better.

Comparison









From the above four graphs, we can clearly visualize that CNN (96.56%) has the highest R Square and Adjacent R2. It shows that the proportion of variation of the Price variable with the independent variable that is explained by the CNN model is good when compared to other ML and DL Techniques. We can also visualize that the R square of Cat Boost (90.56%) and XGB (90.07%) are also good next to the CNN model.

The adjusted R-squared is a modified version of the R-squared statistic that adjusts for the number of predictors in a regression model. The adjacent R2 also good for CNN, then cat boost and XGB.

Conclusion

The top-performing model in our analysis is the Convolutional Neural Network (CNN), which leads with an exceptional R^2 value of 96.56%, underscoring its outstanding performance. Following closely are CatBoost and XGBoost, with R^2 values of 90.56% and 90.07%, respectively.

In comparison to traditional models, Linear Regression demonstrates commendable performance with an R^2 of 89.41%. However, it is noteworthy that advanced models outperform Linear Regression in our study.

When evaluating error metrics, the CNN stands out by recording the lowest Root Mean Squared Error (RMSE) and Mean Squared Error (MSE), signifying a high level of exactness in its predictions. Conversely, the Feed Neural Network exhibits the highest error rates, suggesting a need for optimization in its performance.

Overall insights gleaned from the study underscore the superior capabilities of advanced neural networks, particularly the CNN, in effectively handling complex data. The research emphasizes the transformative potential of machine learning techniques, specifically neural networks, in achieving not only high accuracy but also robust predictions in regression tasks. This information provides valuable insights for understanding the strengths and performance variations among the models considered in the report.

Reference

[1] House Price Index. Federal Housing Finance Agency. https://www.fhfa.gov/ (accessed September 1, 2019).

[2] Fan C, Cui Z, Zhong X. House Prices Prediction with Machine Learning Algorithms. Proceedings of the 2018 10th International Conference on Machine Learning and Computing - ICMLC 2018. doi:10.1145/3195106.3195133.

[3] Phan TD. Housing Price Prediction Using Machine Learning Algorithms: The Case of Melbourne City, Australia. 2018 International Conference on Machine Learning and Data Engineering (ICMLDE) 2018. doi:10.1109/icmlde.2018.00017.

[4] Mu J, Wu F, Zhang A. Housing Value Forecasting Based on Machine Learning Methods. Abstract and Applied Analysis 2014;2014:1–7.doi:10.1155/2014/648047.