Building a Machine Learning Model for the Prediction of Dependent Variables in the Given Datasets

Assignment 1

Machine Learning

IOT Track

Muhammad Farrukh Mehmood

Reg no. 399602 Fall/22

Table of Contents

List of Figures

List of Tables

1 Introduction

The objective of the assignment is to develop a machine learning model to predict the dependent variables (insurance charges and house price) based on the independent variables following datasets.

I. Insurance data set
II. Real estate dataset

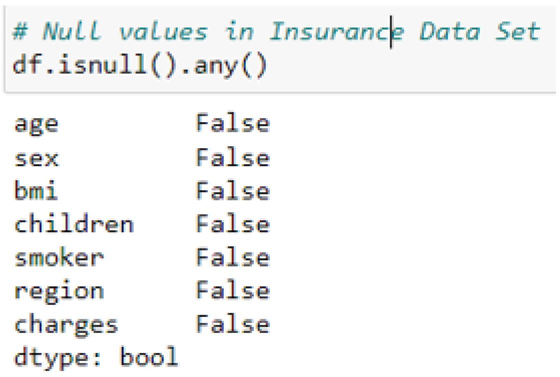
1.1 Road map

To achieve the target following procedural steps were adopted:

- Data cleaning
- Data visualization
- Dividing the data into label(dependent variable) and features (independent variables)
- Splitting the data into training and test sets
- Finding an appropriate model with suitable hyperparameters with the help of GridSearchCV
- Training the model with training data
- Validating the model on train and test sets with different scoring parameters

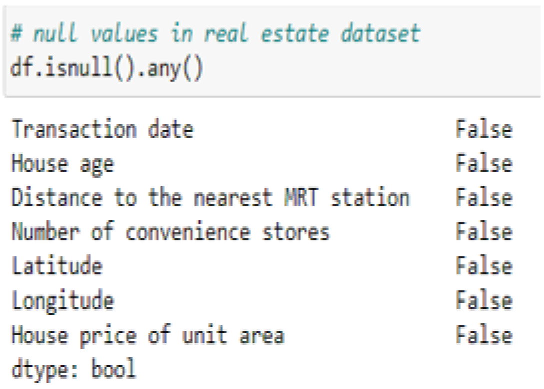
2 Data Cleaning

Firstly, the data were checked for missing or null values. To quantify the null values "isnull().any()" command was used. The command reported no null values in both the data sets.



```
# Null values in Insurance Data Set
df.isnull().any()
```

```
age         False
sex         False
bmi         False
children    False
smoker      False
region      False
charges     False
dtype: bool
```

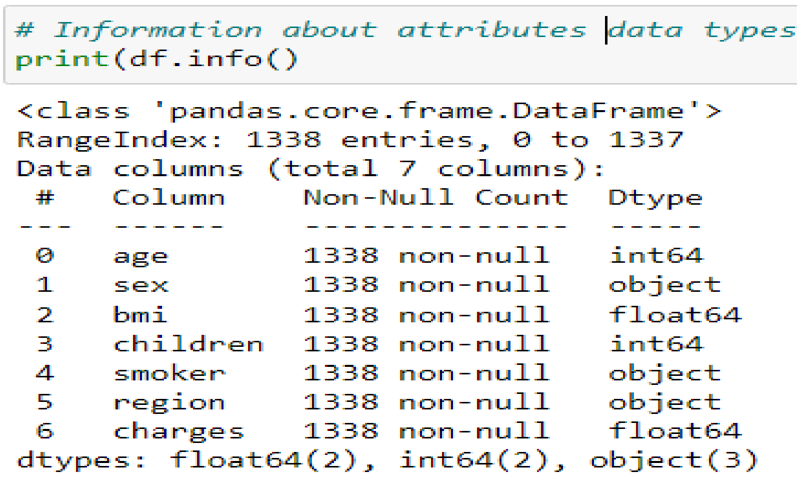


```
# null values in real estate dataset
df.isnull().any()
```

```
Transaction date                        False
House age                               False
Distance to the nearest MRT station     False
Number of convenience stores            False
Latitude                                False
Longitude                               False
House price of unit area                False
dtype: bool
```

Figure 1: Null value reports for insurance and real estate data respectively

Python's 'info()' command provided the information that all the attributes, other than categorical data, contained numerical values.



```
# Information about attributes data types
print(df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
```

Figure 2: Information about the attributes of real estate data

One hot encoding technique was implemented to tackle with categorical data which will be discussed latter.

Python’s ‘missingno’ library was used to visualize the missing data if any. Following charts show that all the features of the data set contain same number of values and no missing data is reported.

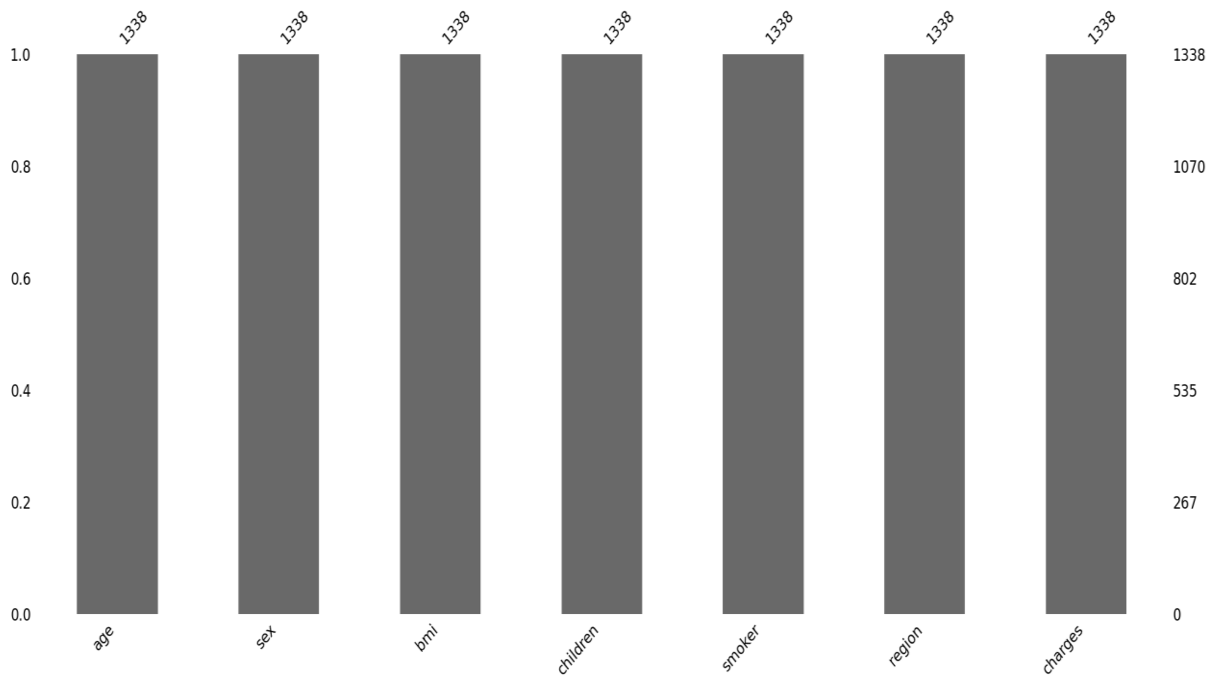


Figure 3: Missing value chart for insurance dataset

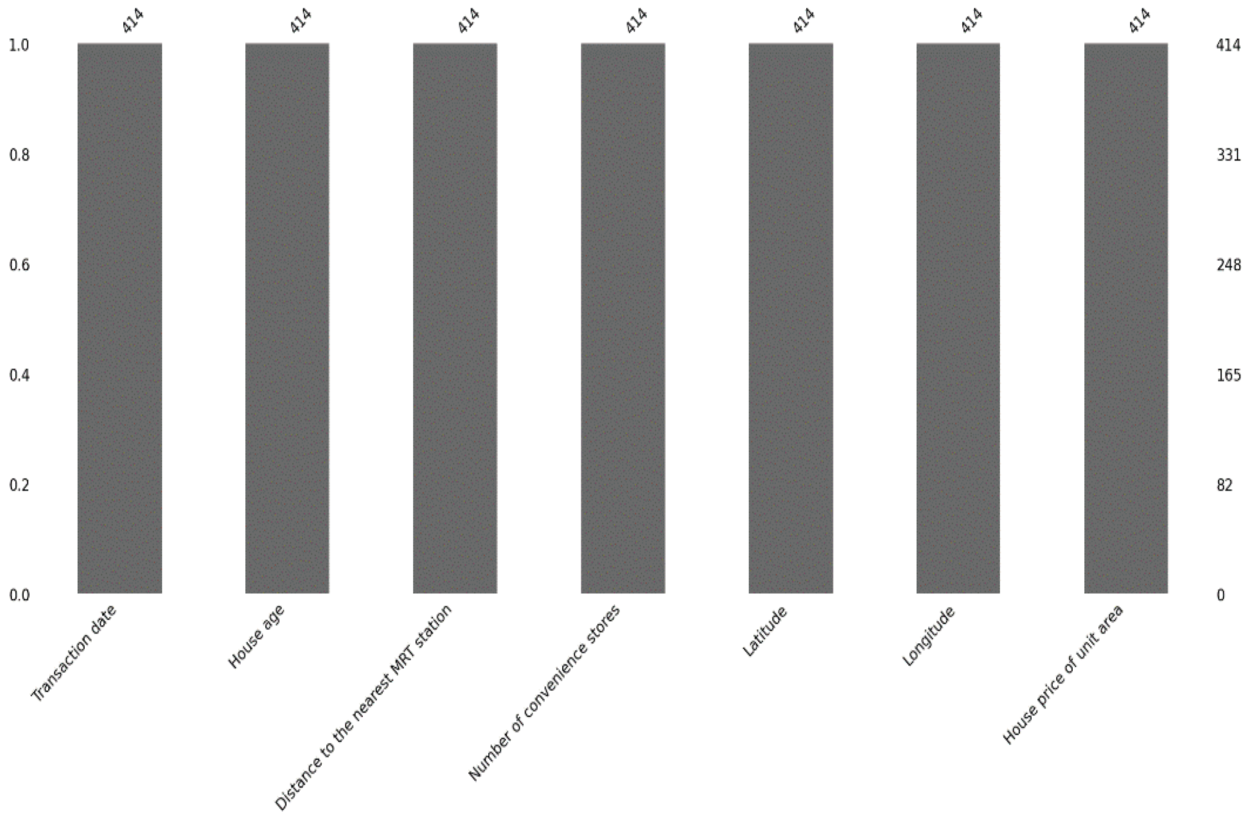


Figure 4: Missing value chart for real estate dataset

3 Data Visualization

3.1 Pair plot

Pair plots were used to quickly visualize patterns and relationships between different features of dataset.

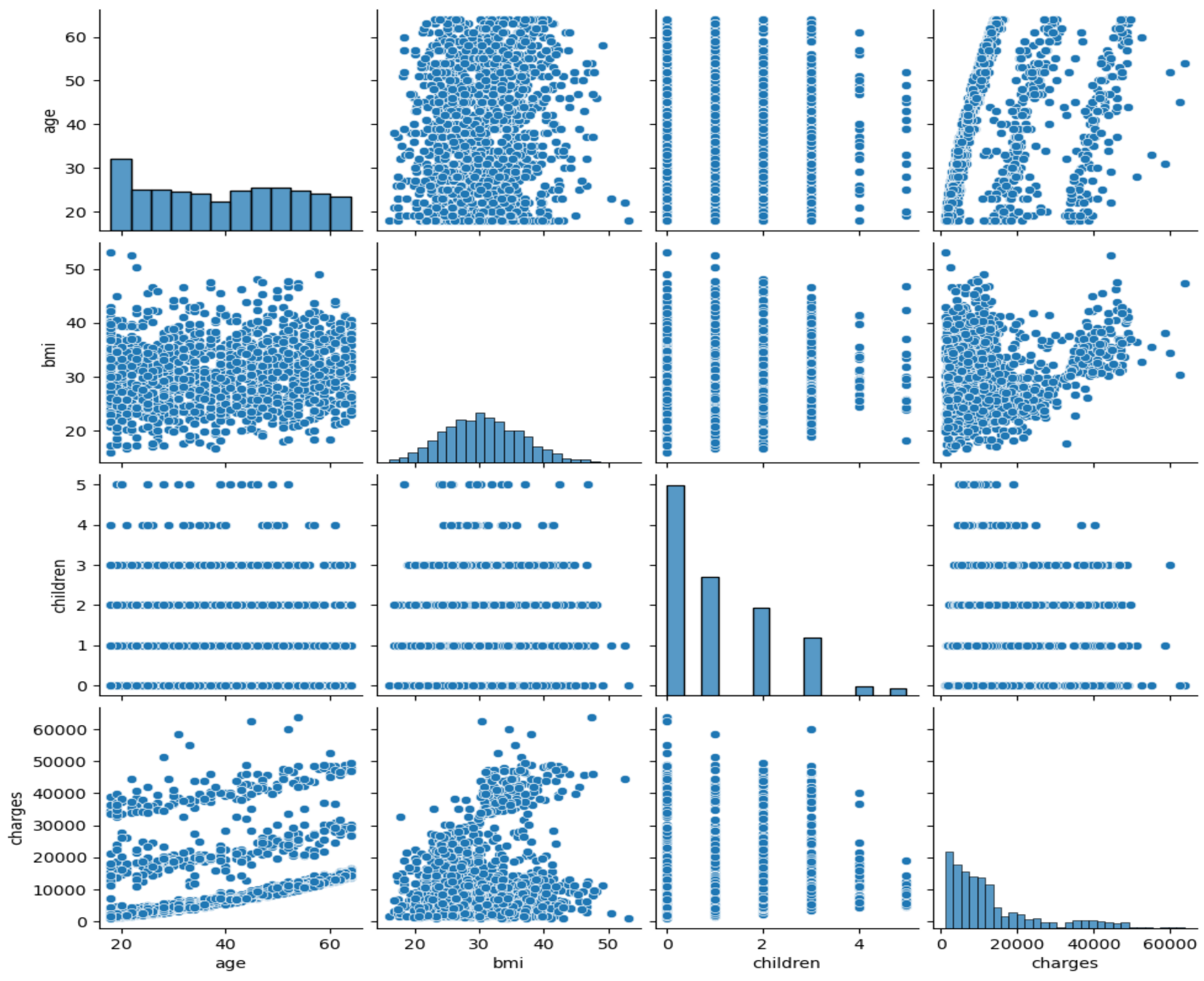


Figure 5: Pair plots for insurance dataset

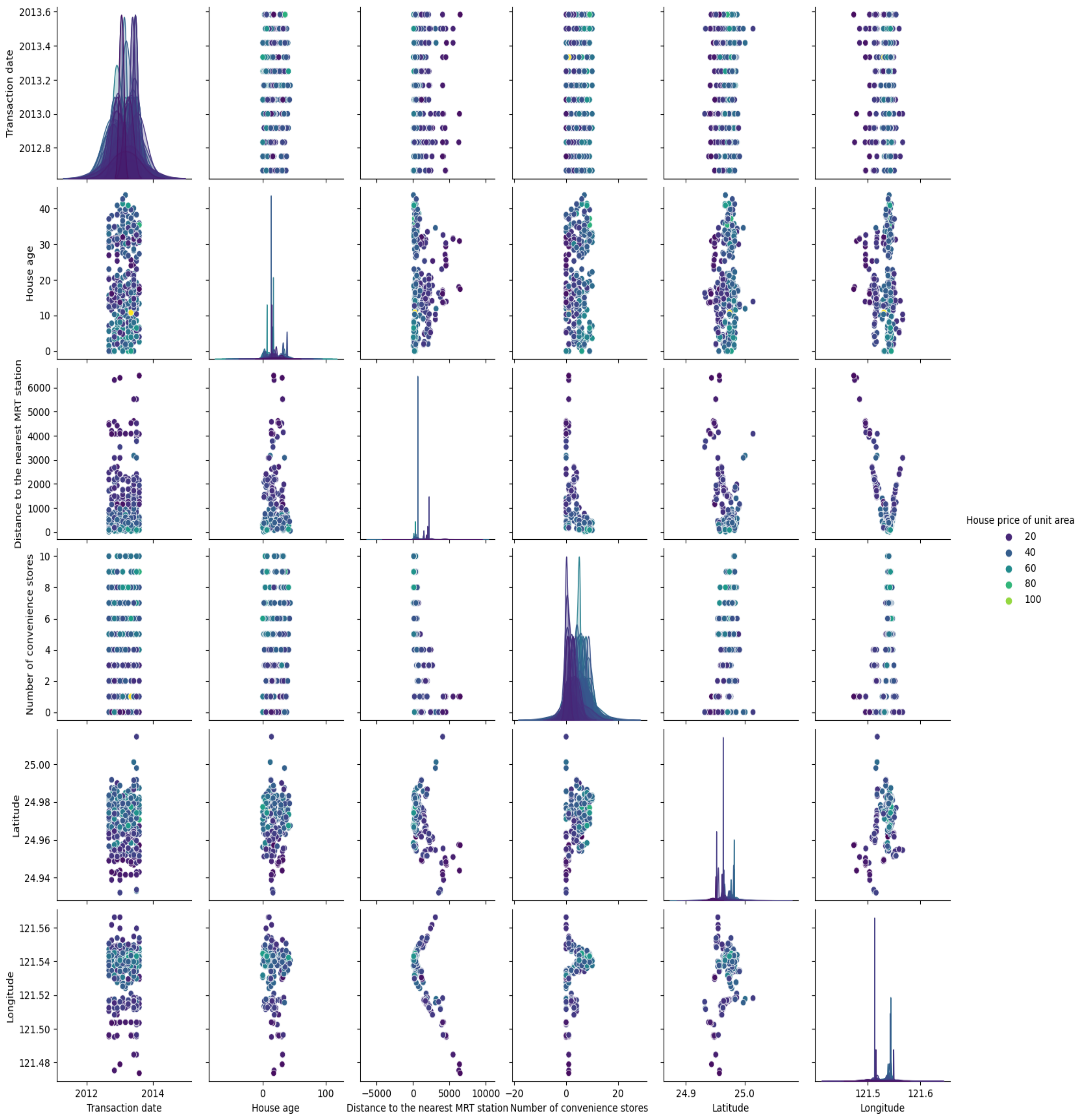


Figure 6: Pair plots for real estate dataset

3.2 Heat map

To identify the correlation between variables of datasets, heatmaps were used.

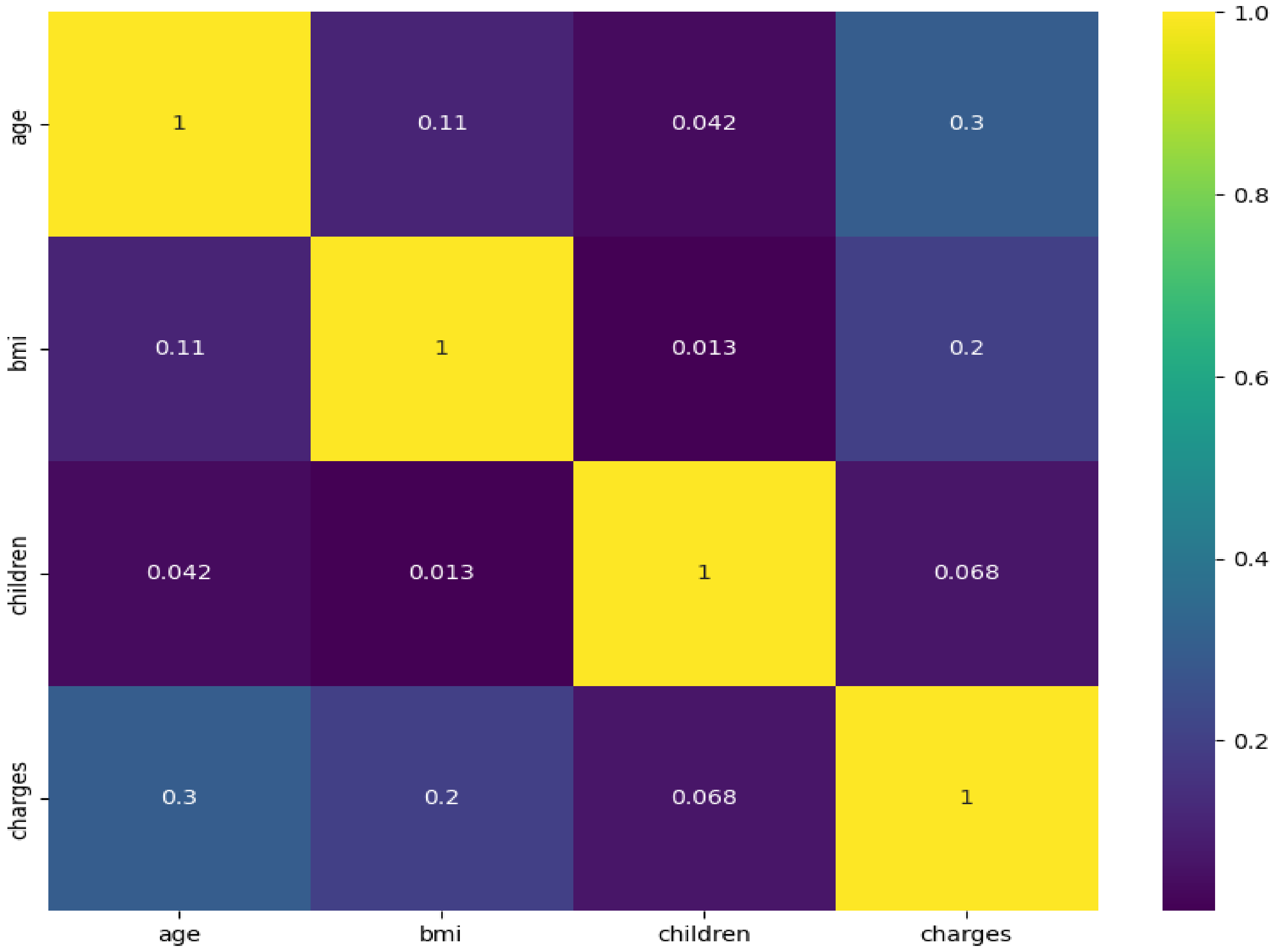


Figure 7: Heatmap for insurance dataset

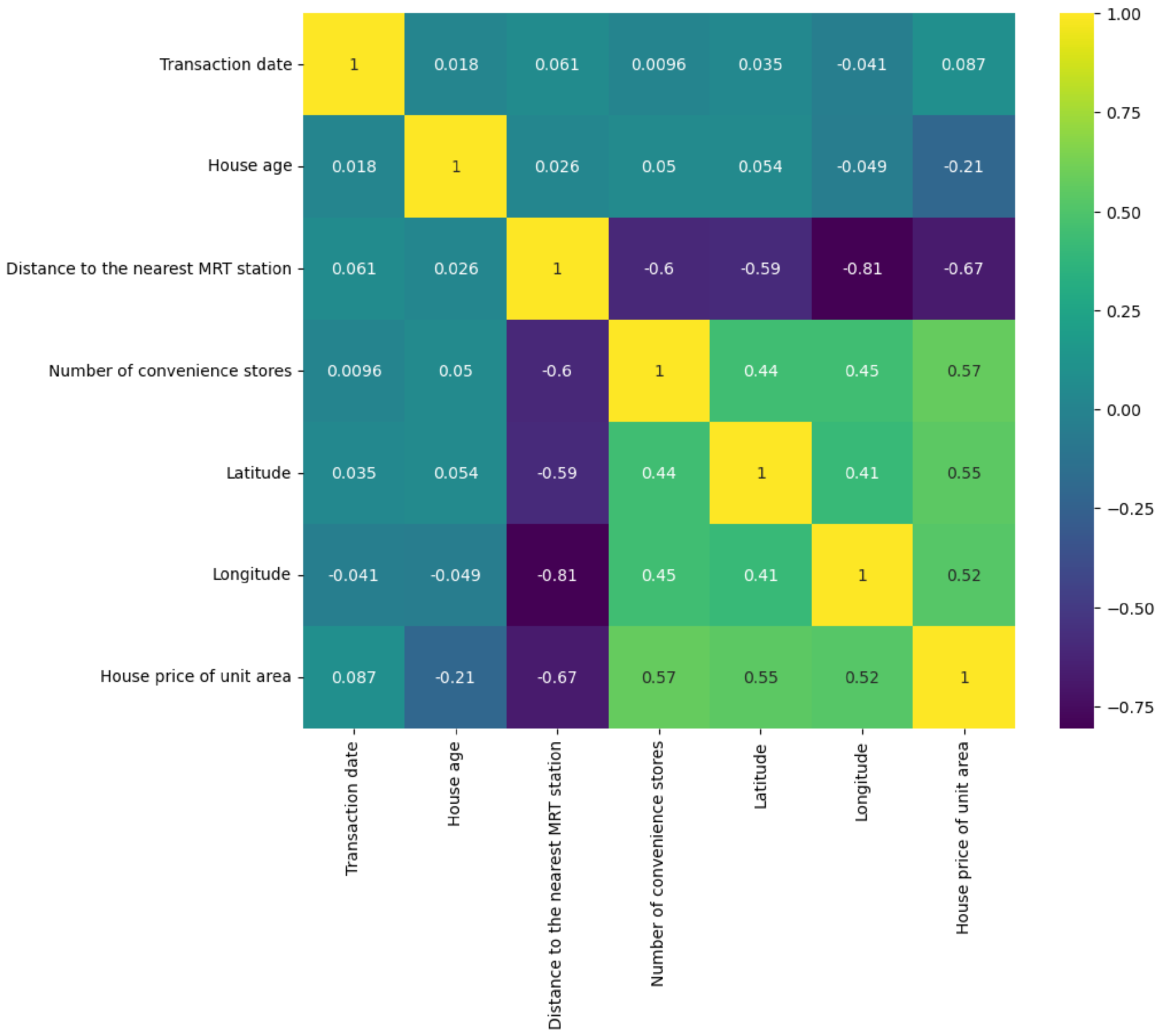


Figure 8: Heatmap for real estate dataset

4 Data preprocessing

4.1 Converting the Categorical Data into numerical

Only Insurance dataset contained the categories data namely ‘sex, smoker, and region’ columns. One Hot Encoding technique was implemented to achieve this target. Following procedure was adopted.

- Dummy variables were created corresponding to the categorical attributes using ‘get_dummies()’ command
- These dummy variables were concatenated with the actual data set.
- The actual categorial attributes columns were dropped off from the data set to get pure numerical dataset

The implementation of the above procedure can be seen in the code.

4.2 Dividing the Actual data into ‘feature’ and ‘label’ datasets

The actual datasets were divided into two new date sets. All the independent variables were assigned a dataset ‘feature’ and all the single dependent variable was assigned the dataset ‘label’.

4.3 Splitting the data into test and train data set

For model building and validation, the data were split in two parts: 70 percent data for model training and 30 percent data for model testing. For this purpose, ‘train_test_split()’ function was used from ‘sklearn’ library.

5 Selection and training the model

Initially sklearn's LinearRegression() module was implemented with StandardScaler normalization technique. The mean absolute error and score on training and test data are give in the following table.

Table 1: Results of LinearRegression model

Insurance Data		
	Training Data	**Test Data**
MAE	4194.3	4194.3
Score	0.744	0.767
Real estate Data		
	Training Data	**Test Data**
MAE	6.09	6.96
Score	0.576	0.579

5.1 Feature Scaling

Next model was tested for following 3 scaling techniques:

- Standard Scalar
- Z score
- Log transform

This time sklearns SGDRegressor model was implemented. Following table contains the errors related to above mentioned normalization techniques.

Table 2: Result of SGDRegressor model with diffenrent data transforms

Insurance Data						
	Standard Scaler		**Z score**		**Log transform**	
	Training Data	**Test Data**	**Training Data**	**Test Data**	**Training Data**	**Test Data**
MAE	4172	4229	4172	4179	4689	4574
Real estate Data						
MAE	6.26	5.87	6.43	6.26	5.846	5.44

It is obvious from the above table that standard scaler transform produces least errors for insurance data and log transform produces least errors for real estate data.

5.2 Hyper parameter tunning

Sklearn contains a bunch of linear regression modules with plenty of hyper parameters. Therefore it becomes nearly impossible to check each module with various hyper parameters to find an optimum model for the given dataset. To ease the difficulty of selecting suitable hyperparameters, sklearn provides a function “GridSearchCV”. GridSearchCV can take a number of models with various values for hyperparameters. It then processes the given data with different combinations of hyperparameters. This enables it to identify the most appropriate model and the optimal hyperparameter values that yield the best performance.

Following is detail of inputs that were given to GridSearchCV:

Estimator:

LinearRegression(), SGDRegressor()

LinearRegression() was set for default values of hyperparameters. Only SGDRegressor() was evaluated for different hyper parameter values.

Table 3: Range of Hyperparameter values for SGDRegressor

Hyper parameter	**Value range**
Feature Scaling	StandardScaler(), log_transform, zscore_transform
Regularization techniques	l1(lasso regression), l2(ridge regression), elasticnet
Regularization parameter	0.0001, 0.001, 0.01, 1
Warm start	True, False
Learning Rate	0.01, 0.001, 0.02
Tolerance	0.001

After processing the above input values, GridSearchCV came up with following suitable values for hyper parameters.

For Insurance data:

Table 4: Hyper parameter values for insurance data

Hyper parameter	**Value**
Feature scaling	StandardScaler()
Estimator	SGDRegressor
Regularization techniques	l1(lasso regression)
Regularization parameter	1
Learning rate	0.001
Warm start	True
Tolerance	1e-3

For real estate data:

Table 5: Hyper parameter values for real estate data

Hyper parameter	**Value**
Feature scaling	StandardScaler()
Estimator	SGDRegressor
Regularization techniques	l2(Ridge regression)
Regularization parameter	0.01
Learning rate	0.02
Warm start	True
Tolerance	1e-3

The detailed implementation of GridsearcCV can be found in the code.

6 Evaluation of hyperparameters obtained through GridsearchCv method

For both the datasets, SGDRegreesor model was found to be the most appropriate. The accuracy of the model was tested based on the hyperparameter values obtained, and various evaluation metrics were used to access its performance.

6.1 Gradient Convergence:

For both the data sets, gradient converged before reaching 50 iterations.

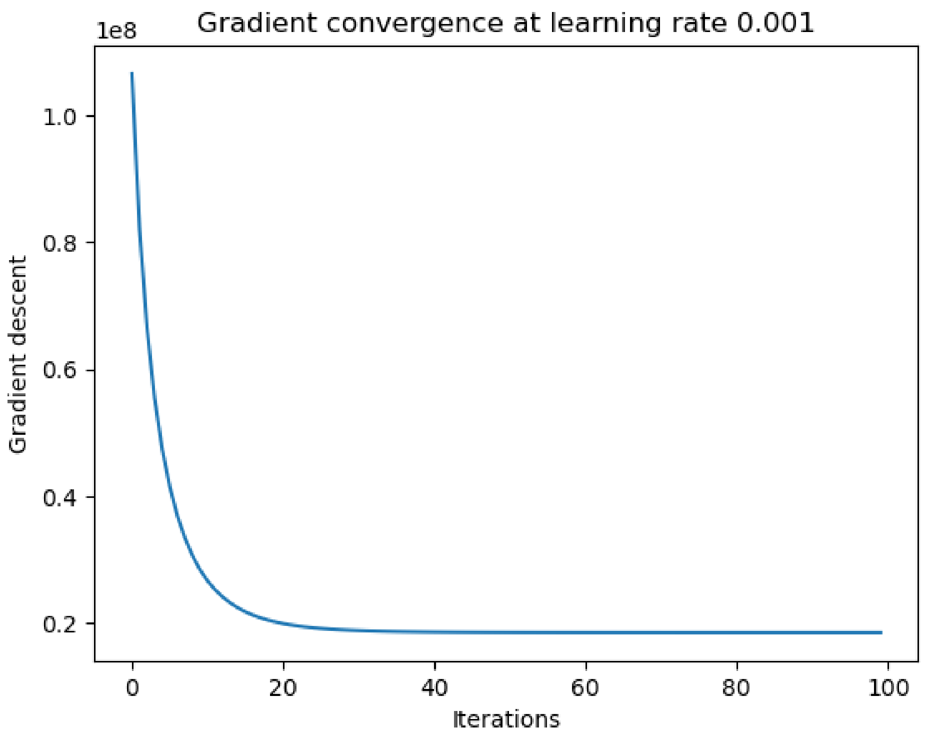


Figure 9: Gradient convergence for insurance data

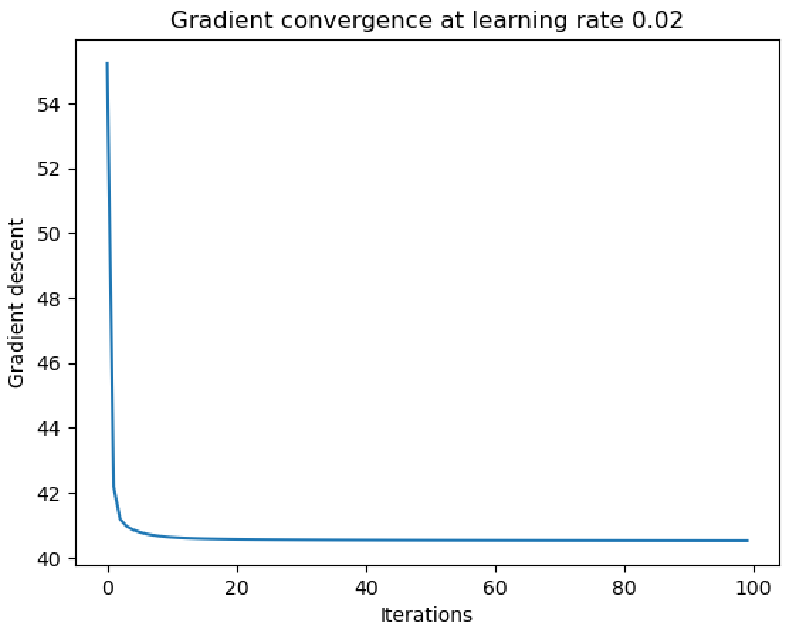


Figure 10: Gradient convergence for real estate data

6.2 Training Loss and accuracy

The training loss was calculated in terms of mean absolute error between the predicted values for labels and actual label values of train portion of datasets.

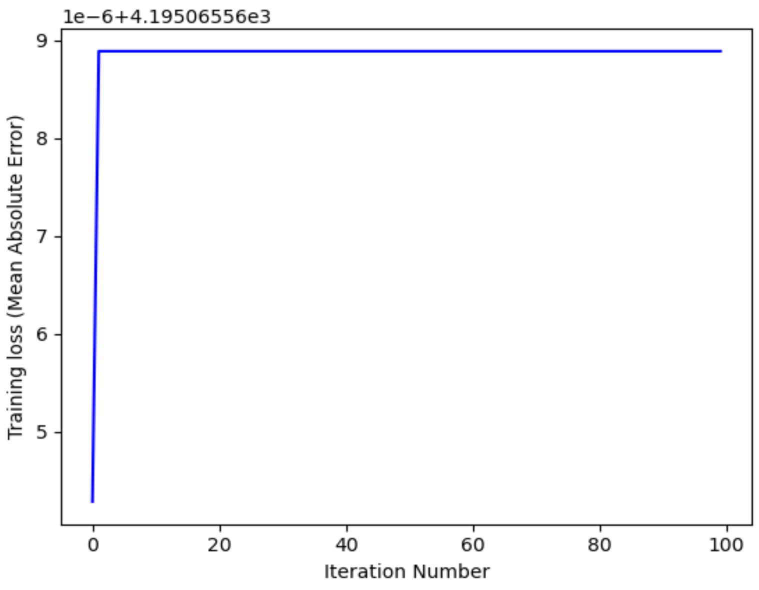


Figure 11: Training loss for insurance data

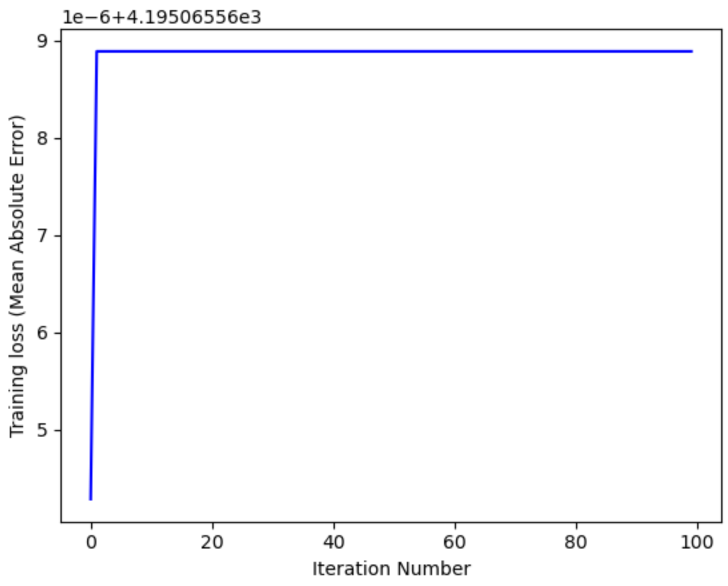


Figure 12: Training loss for real estate data

In case of train data, mean absolute error was 4195 for insurance data and 6.28 for real estate data. The accuracy of model for insurance data remained 0.744 and that for real estate data was 0.57.

6.3 Inference loss and accuracy

The inference loss was calculated in terms of mean absolute error between the predicted values for labels and actual label values test portion of datasets.

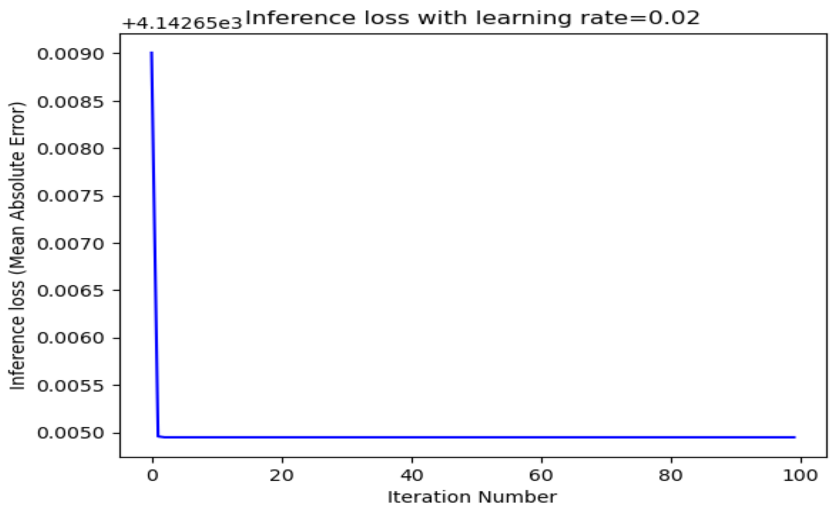


Figure 13: Inference loss for insurance data

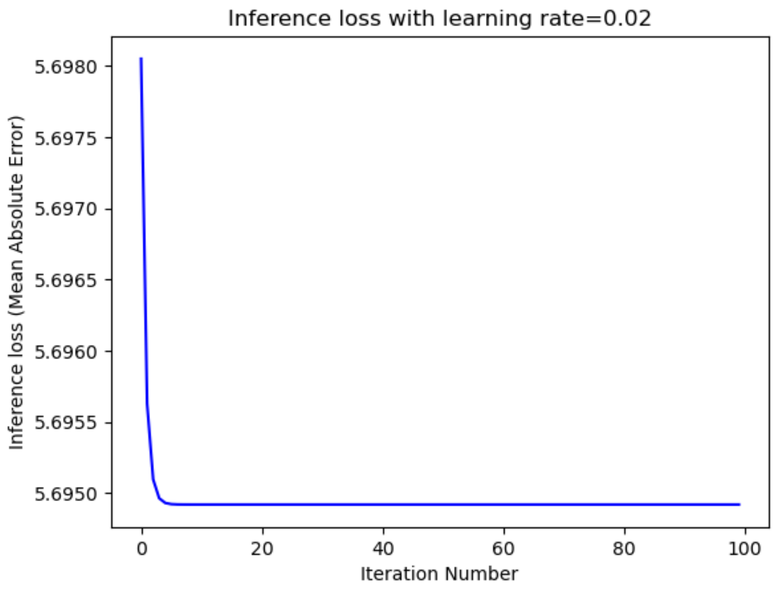


Figure 12: Inference loss for real estate data

For test data, mean absolute error was 4237 for insurance data and 5.6 for real estate data. The accuracy of model for insurance data remained 0.766 and that for real estate data was 0.579.