Numerical Model Document

- Dataset

- Dataset Description: The Ames Housing Dataset is a well-known dataset in the field of machine learning and data analysis. It contains various features and attributes of residential homes in Ames, Iowa, USA. The dataset is often used for regression tasks, particularly for predicting housing prices

    - Number of Instances: The dataset consists of 2,930 instances or observations

    - Number of Features: There are 79 different features or variables that describe various aspects of the residential properties.(We Work With Only 17 Feature)

    - Target Variable: The target variable in the dataset is the "SalePrice," representing the sale price of the houses.

    - Data Types: The features include both numerical and categorical variables

Differences Between Linear Regression and KNN Regression:

1. Algorithm Approach:

- **Linear Regression**:
  Fits a straight-line relationship between the input features and the target variable, assuming the relationship is linear.
- **KNN Regression:**
  Makes predictions based on the average of the target values of the k-nearest neighbors in the feature space, handling non-linear relationships better.

Metric	Linear Regression	KNN Regression
MSE	0.158 (Lower, better fit)	0.175 (higher)
RMSE	0.398	0.418
R^2	0.850	0.8342
Train Accuracy	0.849	0.846
Test Accuracy	0.850 (Lower, generalizers less)	0.8342

R^2 = Test Accuracy

2.Interpretation

- Linear Regression has a lower MSE and higher R^2, indicating it explains the variance of the target variable better for this dataset.
- KNN Regression performs better in Test Accuracy, suggesting it generalizes well for unseen data compared to Linear Regression.

3.Sensitivity to Data Patterns

- **Linear Regression:**
    - Performs better when the relationship between features and target is linear.
    - Struggles with non-linear data and is sensitive to outliers.
- KNN Regression:
    - Handles non-linear relationships effectively by relying on local data patterns.
    - Less sensitive to outliers due to averaging over neighboring data points.

CODE EXPALINTION

Dataset Selection / Importing libraries

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler, LabelEncoder

dataset = pd.read_csv('C:/Users/d_tol/Desktop/L.3 - first term/ML/ML Sections/AmesHousing.csv')
columns = [
        'Lot Frontage','Overall Qual','Year Built','Year Remod/Add','Mas Vnr Area','Exter Qual','BsmtFin SF 1',
        'Total Bsmt SF','1st Flr SF','Gr Liv Area','Full Bath','Kitchen Qual','TotRms AbvGrd','Fireplaces',
        'Garage Yr Blt', 'Garage Cars', 'Garage Area','SalePrice'
        ]

num_columns = [
        'Lot Frontage','Year Built','Year Remod/Add','Mas Vnr Area','BsmtFin SF 1',
        'Total Bsmt SF','1st Flr SF','Gr Liv Area','Full Bath','TotRms AbvGrd','Fireplaces',
        'Garage Yr Blt', 'Garage Cars', 'Garage Area','SalePrice'
        ]

cat_columns = ['Overall Qual','Exter Qual', 'Kitchen Qual']
df = dataset[columns]
print(df.shape)
df.head()
```

The code imports libraries like pandas and scikit-learn, loads data from a CSV file, and prepares data by selecting features and separating numerical and categorical variables. It likely involves splitting data, preprocessing, training models (e.g., linear regression, KNN), and evaluating their performance.

	Lot Frontage	Overall Qual	Year Built	Year Remod/Add	Mas Vnr Area	Exter Qual	BsmtFin SF 1	Total Bsmt SF	1st Flr SF	Gr Liv Area	Full Bath	Kitchen Qual	TotRms AbvGrd	Fireplaces	Garage Yr Blt	Garage Cars	Garage Area	SalePrice
0	141.0	6	1960	1960	112.0	TA	639.0	1080.0	1656	1656	1	TA	7	2	1960.0	2.0	528.0	215000
1	80.0	5	1961	1961	0.0	TA	468.0	882.0	896	896	1	TA	5	0	1961.0	1.0	730.0	105000
2	81.0	6	1958	1958	108.0	TA	923.0	1329.0	1329	1329	1	Gd	6	0	1958.0	1.0	312.0	172000
3	93.0	7	1968	1968	0.0	Gd	1065.0	2110.0	2110	2110	2	Ex	8	2	1968.0	2.0	522.0	244000
4	74.0	5	1997	1998	0.0	TA	791.0	928.0	928	1629	2	TA	6	1	1997.0	2.0	482.0	189900

Dataset information

```
df.info()
```

The code df.info() is used to display a concise summary of a DataFrame, including column names, data types, non-null values, and memory usage.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Lot Frontage    2440 non-null   float64
 1   Overall Qual    2930 non-null   int64
 2   Year Built      2930 non-null   int64
 3   Year Remod/Add  2930 non-null   int64
 4   Mas Vnr Area    2907 non-null   float64
 5   Exter Qual      2930 non-null   object
 6   BsmtFin SF 1    2929 non-null   float64
 7   Total Bsmt SF   2929 non-null   float64
 8   1st Flr SF      2930 non-null   int64
 9   Gr Liv Area     2930 non-null   int64
 10  Full Bath       2930 non-null   int64
 11  Kitchen Qual    2930 non-null   object
 12  TotRms AbvGrd   2930 non-null   int64
 13  Fireplaces      2930 non-null   int64
 14  Garage Yr Blt   2771 non-null   float64
 15  Garage Cars     2929 non-null   float64
 16  Garage Area     2929 non-null   float64
 17  SalePrice       2930 non-null   int64
dtypes: float64(7), int64(9), object(2)
memory usage: 412.2+ KB
```

Detecting Null Values

```
1   print(df.isnull().sum())
```

```
Lot Frontage       490
Overall Qual         0
Year Built           0
Year Remod/Add       0
Mas Vnr Area        23
Exter Qual           0
BsmtFin SF 1         1
Total Bsmt SF        1
1st Flr SF           0
Gr Liv Area          0
Full Bath            0
Kitchen Qual         0
TotRms AbvGrd        0
Fireplaces           0
Garage Yr Blt      159
Garage Cars          1
Garage Area          1
SalePrice            0
dtype: int64
```

Handle Null Values

```
1   df[num_columns] = df[num_columns].fillna(df[num_columns].mean())
```

Handling Outliers

```python
def remove_outliers_iqr(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df

num_columns = [
        'Lot Frontage','Year Built','Year Remod/Add','Mas Vnr Area','BsmtFin SF 1',
        'Total Bsmt SF','1st Flr SF','Gr Liv Area','Full Bath','TotRms AbvGrd','Fireplaces',
        'Garage Yr Blt', 'Garage Cars', 'Garage Area','SalePrice'
        ]
df = remove_outliers_iqr(df, num_columns)
print("Shape after outlier removal:", df.shape)
```

The code defines a function remove_outliers_iqr to remove outliers from a DataFrame using the Interquartile Range (IQR) method. It calculates Q1, Q3, and IQR for each numerical column, determines lower and upper bounds, and filters the DataFrame to exclude values outside these bounds. Finally, it applies this function to specific columns in the DataFrame and prints the new shape.

Scale Numerical Columns and Encode Categorical Columns

```
scaler = StandardScaler()
df[num_columns] = scaler.fit_transform(df[num_columns])

le = LabelEncoder()
df['Exter Qual'] = le.fit_transform(df['Exter Qual'])
df['Kitchen Qual'] = le.fit_transform(df['Kitchen Qual'])
df.head(20)
```

The code scales numerical features using StandardScaler and encodes categorical features using LabelEncoder. It then displays the first 20 rows of the preprocessed DataFrame.

	Lot Frontage	Overall Qual	Year Built	Year Remod/Add	Mas Vnr Area	Exter Qual	BsmtFin SF 1	Total Bsmt SF	1st Flr SF	Gr Liv Area	Full Bath	Kitchen Qual
1	0.993321	5	-0.244027	-1.030369	-0.587375	3	0.179968	-0.408516	-0.656627	-1.301527	-0.927643	4
2	1.069611	6	-0.342842	-1.171931	0.585191	3	1.367616	1.035865	0.832627	-0.155601	-0.927643	2
4	0.535581	5	0.941755	0.715560	-0.587375	3	1.023068	-0.259878	-0.546567	0.638343	0.972348	4
5	0.840741	6	0.974693	0.715560	-0.370233	3	0.529737	-0.266340	-0.553446	0.572181	0.972348	2
6	-1.981988	8	1.073508	0.857122	-0.587375	2	0.566280	1.064947	0.863582	-0.131783	0.972348	2
7	-1.829408	8	0.777063	0.432436	-0.587375	2	-0.355126	0.877532	0.664097	-0.285279	0.972348	2
8	-2.134568	8	0.875878	0.621185	-0.587375	2	2.038442	1.895385	1.819732	0.603939	0.972348	2
9	-0.532479	7	1.007631	0.762747	-0.587375	3	-1.041613	-0.046613	-0.202628	1.101477	0.972348	2
10	0.611871	6	0.810001	0.526811	-0.587375	3	-1.041613	-0.793039	-1.114066	0.707152	0.972348	4

Split The Data into Training and Testing

```python
X = df.drop(columns=['SalePrice'])
y = df['SalePrice']

print(X.shape)
print(y.shape)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

The code snippet splits the DataFrame into features (X) and target variable (y). It then prints the shapes of X and y and splits the data into training and testing sets using train_test_split with a 0.3 test size and a random state of 0.

```
(2210, 17)
(2210,)
```

Getting the best number of Neighbors to train the knn model

```python
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': range(1, 20)}
grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

print("Best n_neighbors:", grid_search.best_params_['n_neighbors'])

```

The code sets up a grid search to find the best n_neighbors parameter for a KNeighborsRegressor model. It defines a parameter grid with values from 1 to 20, creates a GridSearchCV object, fits it to the training data, and prints the best n_neighbors value found.

```
Best n_neighbors: 11
```

Train the models

```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

This code initializes a LinearRegression model and fits it to the training data (X_train, y_train), training the model to learn the relationship between the features and the target variable.

```
knn_model = KNeighborsRegressor(n_neighbors=11)
knn_model.fit(X_train, y_train)
```

This code snippet creates a KNeighborsRegressor model with 11 neighbors and fits it to the training data. This trains the model to predict values based on the nearest neighbors in the training data

Calculate The MSE, RMSE and R2 Score

Linear Regression

```
lr_preds = lr_model.predict(X_test)

lr_mse = mean_squared_error(y_test, lr_preds)
lr_r2 = r2_score(y_test, lr_preds)
lr_rmse = np.sqrt(lr_mse)


print("Linear Regression: MSE =", lr_mse, " , RMSE = ", lr_rmse,", R2 =", lr_r2)
print("Train accuracy =", lr_model.score(X_train, y_train))
print("Test accuracy  =", lr_model.score(X_test, y_test))
```

This code snippet calculates the Mean Squared Error (MSE) and R-squared (R2) scores for a linear regression model on a test set. It also calculates the Root Mean Squared Error (RMSE) and prints the results, along with the model's training and test accuracies.

```
Linear Regression: MSE = 0.1583920102719532  , RMSE =  0.3979849372425459 , R2 = 0.850066309507564
Train accuracy = 0.8489695768305543
Test accuracy  = 0.850066309507564
```

KNN Regression

```python
knn_preds = knn_model.predict(X_test)

knn_mse = mean_squared_error(y_test, knn_preds)
knn_r2 = r2_score(y_test, knn_preds)
knn_rmse = np.sqrt(knn_mse)

print("KNN Regression: MSE =", knn_mse, ", RMSE =", knn_rmse, ", R2 =", knn_r2)
print("Train accuracy =", knn_model.score(X_train, y_train))
print("Test accuracy  =", knn_model.score(X_test, y_test))
```

This code calculates evaluation metrics for a K-Nearest Neighbors (KNN) regression model:

1. It predicts values on the test set using the trained KNN model.
2. It computes the Mean Squared Error (MSE), R-squared (R2), and Root Mean Squared Error (RMSE) between the true and predicted values.
3. It prints the calculated metrics for the KNN model.
4. It also prints the training and testing accuracies of the model.

```
KNN Regression: MSE = 0.17511389315816378 , RMSE = 0.4184661194865885 , R2 = 0.8342373948495132
Train accuracy = 0.8460721935267181
Test accuracy  = 0.8342373948495132
```