BDAT1007 – Group Final Project

# Students Involved:

1. Sarthak Pokharel 200557906
2. Laxman Rokaya 200562874
3. Ashmita Devkota 200574820
4. Yanish Shahi 200571165
5. Nilam Shrestha 200571162
6. Meru Sangroula 200565532

# Business Cases

Some questions to consider:

* **What’s the purpose of your project?**

The purpose of this project is to develop a predictive model for real estate pricing based on various property features such as area, bedrooms, bathrooms, etc.

* **What is the use case?**

Assisting real estate investors, buyers, and sellers in making informed decisions about property pricing, investment opportunities, and market trends.

* **Who are your audiences?**

o Real estate investors: Provide insights into property valuation, investment potential, and market trends to guide investment decisions.

o Homebuyers and sellers: Offer guidance on property pricing, market trends, and factors influencing property values to aid in buying or selling decisions.

o Real estate agents and agencies: Support in property valuation, market analysis, and client consultations for better client service and decision-making.

# Technical Design Document

* **How do you collect the data?**

The data is collected form Kaggle.

**Link to dataset**: https://www.kaggle.com/datasets/yasserh/housing-prices-dataset

* **What’s the toolset / coding language(s) used?**

Python (Flask) has been used for this project alongside HTML, CSS and Javascript.

* **How do the data operators & data models look like?**

**Data Operators:**

**Data Cleaning Operators:** Preprocessing steps such as handling missing values, encoding categorical variables, and scaling numerical features fall under this category.

**Machine Learning Operators:** Training and evaluating machine learning models such as Gaussian Naive Bayes and Random Forest Regressor.

**Data Models:**

**Gaussian Naive Bayes Model:** Used for predicting housing prices based on various features.

**Random Forest Regressor Model:** Also used for predicting housing prices but with a different algorithm and potentially different features' importance.

* **What does the data flow & data pipeline look like? Any processing mapping / flow chart?**

**Data Flow:**

**Data Preprocessing:** Read housing data from a CSV file (Housing.csv). Clean and transform data using functions like preprocess\_data.

**Model Training:** Training machine learning models like Gaussian Naive Bayes and Random Forest Regressor.

**Model Evaluation:** Evaluating model performance using metrics such as RMSE (Root Mean Squared Error).

**Visualization:** Generating plots and visualizations using Plotly to understand data correlations, clustering, and model predictions.

**Data Pipeline:**

**Step 1:** Get data from Housing.csv.

**Step 2:** Preprocess the data (cleaning, encoding, scaling) using preprocess\_data function.

**Step 3:** Perform exploratory data analysis (EDA) and visualize correlations using get\_correlation\_heatmap.

**Step 4:** Cluster data using KMeans clustering with get\_clustered\_data.

**Step 5:** Determine optimal cluster number using the Elbow method with get\_elbow\_plot.

**Step 6:** Train and evaluate Gaussian Naive Bayes model using train\_and\_plot\_model.

**Step 7:** Train and evaluate Random Forest Regressor model using train\_and\_plot\_rf\_model.

**Step 8:** Visualize model predictions and feature importance.

**Models Used:**

**Random Forest Regressor (rtrain\_and\_plot\_model function):**

Random Forest is a powerful ensemble learning technique that works well for regression tasks like predicting house prices. Here are the reasons why Random Forest is a good choice:

1. Handles Non-linearity: Random Forest can capture non-linear relationships between features and the target variable, which is important for real-world datasets where relationships are rarely linear.
2. Handles Mixed Data Types: It can handle both numerical and categorical features without requiring extensive preprocessing. This is beneficial as your dataset contains a mix of numerical and categorical features.
3. Feature Importance: Random Forest provides feature importances, helping you understand which features are most influential in predicting house prices. This information can be valuable for feature selection and understanding the dataset.

**Gaussian Naive Bayes Regressor (train\_and\_plot\_model function):**

As to why Gaussian Naive Bayes is chosen:

1. **Handles Categorical Features**: It works well with categorical features, which is relevant as your dataset includes categorical variables like mainroad, guestroom, etc.
2. **Simplicity and Speed:** Naive Bayes is computationally efficient and easy to implement, making it suitable for quick model prototyping and testing.

**TPOT Regressor (automate\_model function):**

Why TPOT has been used

1. Automated Model Selection: TPOT searches for the best combination of preprocessing steps and machine learning algorithms for your dataset. This saves time and effort compared to manually trying out different models.
2. Optimizes Performance: By exploring various pipelines and model configurations, TPOT aims to find the model that performs best on your dataset in terms of predictive accuracy (RMSE in your case).
3. Handles Complex Data: TPOT can handle datasets with mixed data types, feature scaling, missing values, etc., providing a robust solution for model building.

**Data Type:** The datatype we are working with is structured data.

# Grading components:

* Originality – WARNING: Plagiarism is a serious academic offence!!!
* Amount of effort you put on the research and design.
* Creativity
* Complexity of the project. Some factors to consider:
  + **Data size – we are working with the big data world**
  + **Data type – are you working with structured data, semi-structured data, unstructured data?**
  + **Any data preprocessing & transformation?**
  + **Any data visualization? How do you tell a story from the data?**
  + **Any / How many machine learning model(s) are used? And why they are used?**
  + **Any feature like Auto-Model, Turbo-Prep, etc. being used?**
  + **What’s your findings? How do you explain the result?**

Appendix (code)

from flask import Flask, render\_template

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.cluster import KMeans

import numpy as np

import plotly.express as px

import plotly.graph\_objs as go

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from tpot import TPOTRegressor

app = Flask(\_\_name\_\_, template\_folder='templates/html/')

def preprocess\_data(df):

    df.dropna(inplace=True)

    le = LabelEncoder()

    categorical\_cols = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatus']

    df[categorical\_cols] = df[categorical\_cols].apply(le.fit\_transform)

    scaler = StandardScaler()

    numerical\_cols = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']

    df[numerical\_cols] = scaler.fit\_transform(df[numerical\_cols])

    return df

def calculate\_inertia(data, max\_k=10):

    inertia\_values = []

    for k in range(1, max\_k + 1):

        kmeans = KMeans(n\_clusters=k, random\_state=42)

        kmeans.fit(data)

        inertia\_values.append(kmeans.inertia\_)

    return inertia\_values

def get\_correlation\_heatmap(df):

    corr\_matrix = df.corr()

    fig = px.imshow(corr\_matrix,

                    labels=dict(color="Correlation"),

                    x=corr\_matrix.index,

                    y=corr\_matrix.columns,

                    color\_continuous\_scale='sunset')

    return fig.to\_json()

def get\_clustered\_data(df, numerical\_cols):

    kmeans = KMeans(n\_clusters=3, random\_state=42)

    df['cluster'] = kmeans.fit\_predict(df[numerical\_cols])

    fig = px.scatter(df, x='area', y='price', color='cluster', title='Clustered Data')

    return fig.to\_json()

def get\_elbow\_plot(df, numerical\_cols):

    inertia\_values = calculate\_inertia(df[numerical\_cols])

    k\_values = np.arange(1, len(inertia\_values) + 1)

    fig = go.Figure()

    fig.add\_trace(go.Scatter(x=k\_values, y=inertia\_values, mode='lines+markers'))

    fig.update\_layout(title='Elbow Method for Optimal K',

                      xaxis\_title='Number of Clusters (K)',

                      yaxis\_title='Inertia')

    return fig.to\_json()

# Function to train Gaussian Naive Bayes regressor, make predictions, and plot results

def train\_and\_plot\_model(df):

    # Split the data into features and target variable

    X = df.drop(['price'], axis=1)

    y = df['price']

    # 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)

    # Standardize numerical features

    scaler = StandardScaler()

    X\_train\_scaled = scaler.fit\_transform(X\_train)

    X\_test\_scaled = scaler.transform(X\_test)

    # Train a Gaussian Naive Bayes regressor

    gnb = GaussianNB()

    gnb.fit(X\_train\_scaled, y\_train)

    # Make predictions

    y\_pred = gnb.predict(X\_test\_scaled)

    # Evaluate the model using RMSE

    mse = mean\_squared\_error(y\_test, y\_pred)

    rmse = np.sqrt(mse)

    # Plot actual vs predicted prices using Plotly

    fig = go.Figure()

    fig.add\_trace(go.Scatter(x=y\_test, y=y\_pred, mode='markers', name='Predicted Prices'))

    fig.add\_trace(go.Scatter(x=y\_test, y=y\_test, mode='lines', name='Actual Prices', line=dict(color='red', dash='dash')))

    fig.update\_layout(title='Actual Prices vs Predicted Prices',

                      xaxis\_title='Actual Prices',

                      yaxis\_title='Predicted Prices')

    plot\_json = fig.to\_json()

    return rmse, plot\_json

# Function to train Random Forest regressor, make predictions, and plot results

def rtrain\_and\_plot\_model(df):

    # Split the data into features and target variable

    X = df.drop(['price'], axis=1)

    y = df['price']

    # 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)

    # Train a Random Forest regressor

    rf\_regressor = RandomForestRegressor(random\_state=42)

    rf\_regressor.fit(X\_train, y\_train)

    # Make predictions

    y\_pred = rf\_regressor.predict(X\_test)

    # Evaluate the model using RMSE

    mse = mean\_squared\_error(y\_test, y\_pred)

    rmse = np.sqrt(mse)

    # Get feature importances from the trained model

    importances = rf\_regressor.feature\_importances\_

    features = X.columns

    importance\_df = pd.DataFrame({'Feature': features, 'Importance': importances})

    importance\_df = importance\_df.sort\_values(by='Importance', ascending=False)

    # Plot feature importances

    fig\_importance = go.Figure()

    fig\_importance.add\_trace(go.Bar(x=importance\_df['Feature'], y=importance\_df['Importance'],

                                    marker\_color='skyblue'))

    fig\_importance.update\_layout(title='Feature Importance',

                                 xaxis\_title='Features',

                                 yaxis\_title='Importance')

    # Plot actual vs predicted prices using Plotly

    fig\_predictions = go.Figure()

    fig\_predictions.add\_trace(go.Scatter(x=y\_test, y=y\_pred, mode='markers', name='Predicted Prices'))

    fig\_predictions.add\_trace(go.Scatter(x=y\_test, y=y\_test, mode='lines', name='Actual Prices',

                                         line=dict(color='red', dash='dash')))

    fig\_predictions.update\_layout(title='Actual Prices vs Predicted Prices',

                                  xaxis\_title='Actual Prices',

                                  yaxis\_title='Predicted Prices')

    return rmse, fig\_predictions.to\_json(), fig\_importance.to\_json()

def automate\_model(df):

    X = df.drop(['price'], axis=1)

    y = df['price']

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    # Create a TPOT AutoML instance

    tpot = TPOTRegressor(generations=5, population\_size=50, verbosity=2, random\_state=42)

    # Fit TPOT on the training data

    tpot.fit(X\_train, y\_train)

    # Evaluate the model on the test data

    y\_pred = tpot.predict(X\_test)

    # Evaluate the model using RMSE

    mse = mean\_squared\_error(y\_test, y\_pred)

    rmse = np.sqrt(mse)

    return rmse

@app.route('/')

def index():

    df = pd.read\_csv("Housing.csv")

    df = preprocess\_data(df)

    numerical\_cols = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']

    heatmap\_json = get\_correlation\_heatmap(df)

    clustered\_data\_json = get\_clustered\_data(df, numerical\_cols)

    elbow\_plot\_json = get\_elbow\_plot(df, numerical\_cols)

    rmse, naive\_bayes\_plot\_json = train\_and\_plot\_model(df)

    rmsee, a, importance\_plot\_json = rtrain\_and\_plot\_model(df)

    # Automate model training and evaluation

    rmse\_automl = automate\_model(df)

    return render\_template('index.html', heatmap\_json=heatmap\_json, clustered\_data\_json=clustered\_data\_json,

                           elbow\_plot\_json=elbow\_plot\_json, naive\_bayes\_plot\_json=naive\_bayes\_plot\_json,rmse=rmse,

                           importance\_plot\_json=importance\_plot\_json, rmse\_automl=rmse\_automl)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)