PROJECT – House-Price Prediction

Dataset - https://drive.google.com/file/d/1EMAhSRJ054K6_auHhowHB9V_WkQl4ldY/view?usp=sharing

CODE:-

Backend - main.py File

```
from sklearn.preprocessing import  OneHotEncoder
from sklearn.impute import SimpleImputer
from scipy.stats import mstats
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import pandas as pd
from sklearn.linear_model import  Ridge
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import pickle

df = pd.read_csv("data.csv")

ndf= df.select_dtypes(include=['number'])

cm = ndf.corr()
print(cm)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

df.drop(columns=['date', 'street', 'statezip', 'country'], inplace=True)

print(df)

ncols = df.select_dtypes(include=['number']).columns
im = SimpleImputer(strategy='mean')
df[ncols] = im.fit_transform(df[ncols])
print(df)

df[ncols] = mstats.winsorize(df[ncols].values, limits=[0.05, 0.05])
print(df)

encode = OneHotEncoder(drop='first')
cityenco = encode.fit_transform(df[['city']])
print(df)

print(df.info())

ndf= df.select_dtypes(include=['number'])
cm = ndf.corr()
print(cm)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
```

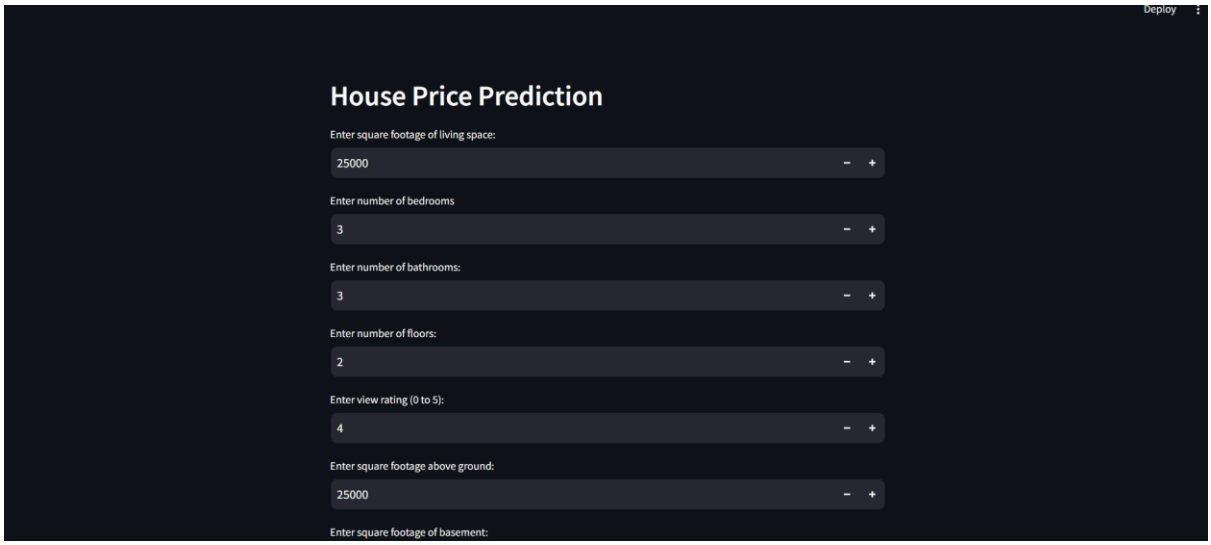
```
plt.show()

X = df[['bedrooms', 'bathrooms', 'floors', 'view', 'sqft_above',
'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living']].values
y = df[['price']]
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

# Linear Regression
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
mse_lr = mean_squared_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mse_lr)

# Ridge Regression
alpha = 1.0  # Regularization strength
model_ridge = Ridge(alpha=alpha)
model_ridge.fit(X_train, y_train)
y_pred_ridge = model_ridge.predict(X_test)
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
rmse_ridge = np.sqrt(mse_ridge)


# Gradient Boosting Regression
model_gb = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
max_depth=3, random_state=0)
model_gb.fit(X_train, y_train.values.ravel())
y_pred_gb = model_gb.predict(X_test)
mae_gb = mean_absolute_error(y_test, y_pred_gb)
mse_gb = mean_squared_error(y_test, y_pred_gb)
rmse_gb = np.sqrt(mse_gb)


# Plotting
info_dict = {'Title': 'House Prices', 'X_label': 'Property Size (sq ft)',
'y_label': 'Price (AUD)'}

# Extract the values from the DataFrame y_test
y_test_values = y_test.values.flatten()

# Plotting
plt.scatter(X_test[:, 8], y_test_values, color='blue', label='Actual data')
plt.plot(X_test[:, 8], y_pred_lr, color='red', label='Linear Regression')
plt.plot(X_test[:, 8], y_pred_ridge, color='green', label='Ridge
Regression')
plt.plot(X_test[:, 8], y_pred_gb, color='purple', label='Gradient Boosting
Regression')
plt.title(info_dict['Title'])
plt.xlabel(info_dict['X_label'])
plt.ylabel(info_dict['y_label'])
plt.legend()
plt.show()

# Print evaluation metrics
print("Linear Regression:")
print("  Mean Absolute Error:", mae_lr)
print("  Root Mean Squared Error:", rmse_lr)
```

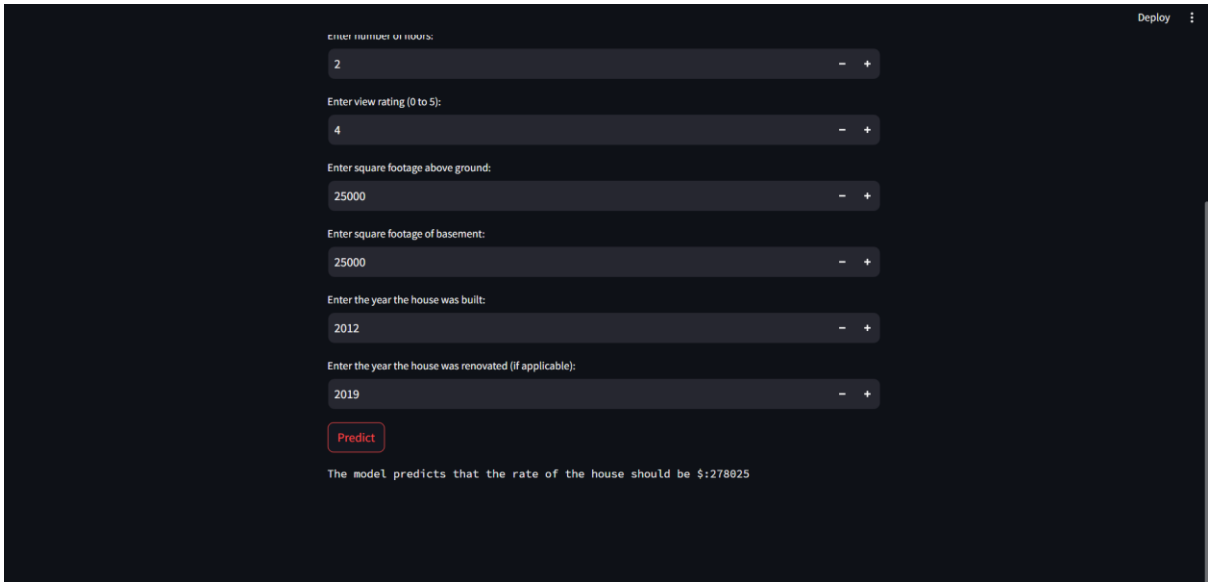
```
print("Ridge Regression:")
print("  Mean Absolute Error:", mae_ridge)
print("  Root Mean Squared Error:", rmse_ridge)
print("Gradient Boosting Regression:")
print("  Mean Absolute Error:", mae_gb)
print("  Root Mean Squared Error:", rmse_gb)
models = {
    "Linear Regression": model_lr,
    "Ridge Regression": model_ridge,
    "Gradient Boosting Regression": model_gb
}

rmse_values = {
    "Linear Regression": rmse_lr,
    "Ridge Regression": rmse_ridge,
    "Gradient Boosting Regression": rmse_gb
}

# Find the model with the lowest RMSE
best_model_name = min(rmse_values, key=rmse_values.get)
best_model = models[best_model_name]
best_model_path = f"best_model.pkl"

# Save the best model
with open(best_model_path, 'wb') as file:
    pickle.dump(best_model, file)



sqft_living= float(input('Enter square footage of living space: '))
bedrooms = int(input('Enter number of bedrooms: '))
bathrooms = float(input('Enter number of bathrooms: '))
floors = float(input('Enter number of floors: '))
view = float(input('Enter view rating (if applicable): '))
sqft_above = float(input('Enter square footage above ground: '))
sqft_basement = float(input('Enter square footage of basement: '))
yr_built = float(input('Enter the year the house was built: '))
yr_renovated = float(input('Enter the year the house was renovated (if
applicable): '))


input_data = pd.DataFrame ({
    'sqft_living': [sqft_living],
    'bedrooms': [bedrooms],
    'bathrooms': [bathrooms],
    'floors': [floors],
    'view': [view],
    'sqft_above': [sqft_above],
    'sqft_basement': [sqft_basement],
    'yr_built': [yr_built],
    'yr_renovated': [yr_renovated]
})

# Predict using the final model
predicted_price = model_gb.predict(input_data)[0]
print("The Predicted price for the house is",predicted_price)
```

Frontend – app.py

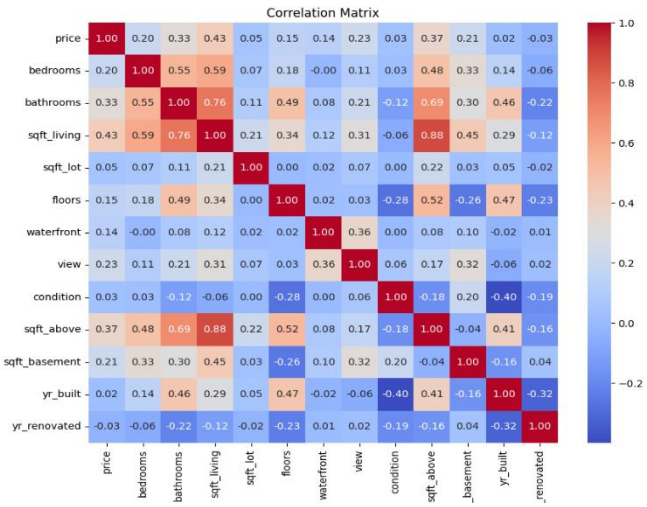
```
import pickle
import streamlit as st
import pandas as pd

st.header("House Price Prediction")
best_model = pickle.load(open("best_model.pkl", 'rb'))

sqft_living = st.number_input("Enter square footage of living space:",
value=0, step=1)
bedrooms = st.number_input("Enter number of bedrooms", value=0, step=1)
bathrooms = st.number_input("Enter number of bathrooms:", value=0, step=1)
floors = st.number_input("Enter number of floors:", value=0, step=1)
view = st.number_input("Enter view rating (0 to 5):",  min_value=0,
max_value=5, step=1)
sqft_above = st.number_input("Enter square footage above ground:", value=0,
step=1)
sqft_basement = st.number_input("Enter square footage of basement:",
value=0, step=1)
yr_built = st.number_input("Enter the year the house was built:", value=0,
step=1)
yr_renovated = st.number_input("Enter the year the house was renovated (if
applicable):", value=0, step=1)

input_data = pd.DataFrame ({
    'sqft_living': [sqft_living],
    'bedrooms': [bedrooms],
    'bathrooms': [bathrooms],
    'floors': [floors],
    'view': [view],
    'sqft_above': [sqft_above],
    'sqft_basement': [sqft_basement],
    'yr_built': [yr_built],
    'yr_renovated': [yr_renovated]
})


if(st.button("Predict")):
    prediction = best_model.predict(input_data)
    st.text(f"The model predicts that the rate of the house should be
$:{prediction}")
```

OUTPUT: -

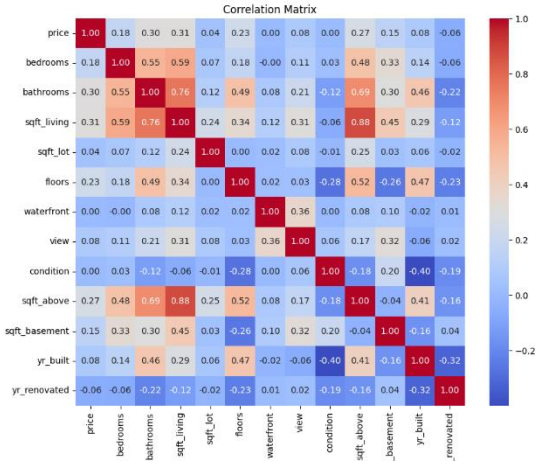




Graphs



Correlation Matrix before Pre-Processing



Co-relation Matrix after pre-processing

Graphs of different model