Source code

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from io import StringIO

# Set up the app
st.title("Housing Data Analysis Dashboard")

# Load data once and store in session_state
if "data" not in st.session_state:
    st.session_state.data = pd.read_csv("housing.csv")

data = st.session_state.data  # always work on this reference

options = st.selectbox(
    "Select processing",
    ["Data Cleaning", "Data Visualization", "Logistic Regression Analysis(ML model)"]
)
```

```
if options == 'Data Cleaning':
    # Data Cleaning Section
    st.header("1. Data Cleaning")

    st.subheader("Raw Data Preview")
    st.write(data.head())
    st.write("Shape:", data.shape)
    st.write("Description:")
    st.write(data.describe())
    st.subheader("Data Information")
    buffer = StringIO()
    data.info(buf=buffer)
    info_str = buffer.getvalue()
    st.text(info_str)

    # Encoding categorical columns
    if st.checkbox("Encoding object columns"):
        cat_cols = data.select_dtypes(include=['object']).columns
        if len(cat_cols) > 0:
            for col in cat_cols:
                le = LabelEncoder()
                data[col] = le.fit_transform(data[col])
            st.success("Label encoding applied.")

            # Show updated info after encoding
```

```
        buffer = StringIO()

        data.info(buf=buffer)

        info_str = buffer.getvalue()

        st.text(info_str)

    else:

        st.info("No categorical columns to encode.")

# Missing values

st.subheader("Missing Values")

st.write(data.isnull().sum())

if st.checkbox("Handle missing values"):

    for col in data.columns:

        if data[col].isnull().sum() > 0:

            data[col].fillna(data[col].mean(), inplace=True)

    st.success("Missing values handled!")

    st.write("Remaining missing values:", data.isnull().sum())


# Duplicate values

st.subheader("Duplicate Values")

st.write(f"Number of duplicates: {data.duplicated().sum()}")

if st.checkbox("Handle duplicated values"):

    data.drop_duplicates(inplace=True)

    st.success("Duplicate rows removed!")

    st.write(f"Remaining duplicates: {data.duplicated().sum()}")


# Outlier detection

st.subheader("Check Outliers (IQR method)")
```

```
  outlier_counts = {}
  numeric_cols = data.select_dtypes(include=['int64', 'float64']).columns
  for col in numeric_cols:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outlier_counts[col] = ((data[col] < lower_bound) |
                (data[col] > upper_bound)).sum()
  st.write(pd.DataFrame(list(outlier_counts.items()), columns=["Column", "Outlier
Count"]))

  # Handle outliers by capping
  def handle_outliers(df):
    for col in df.select_dtypes(include=['int64', 'float64']).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[col] = np.where(df[col] < lower_bound, lower_bound,
                np.where(df[col] > upper_bound, upper_bound, df[col]))
    return df

  if st.checkbox("Handle outliers"):
```

```
        data = handle_outliers(data)

        st.success("Outliers handled (capped to IQR limits)!")

        # Recalculate and show new counts

        new_outlier_counts = {}

        for col in numeric_cols:

          Q1 = data[col].quantile(0.25)

          Q3 = data[col].quantile(0.75)

          IQR = Q3 - Q1

          lower_bound = Q1 - 1.5 * IQR

          upper_bound = Q3 + 1.5 * IQR

          new_outlier_counts[col] = ((data[col] < lower_bound) |

                        (data[col] > upper_bound)).sum()

        st.write(pd.DataFrame(list(new_outlier_counts.items()), columns=["Column", "Outlier
Count"]))


elif options == 'Data Visualization':

  # Data Visualization Section

  st.header("2. Data Visualization")


  # Select visualization type

  viz_type = st.selectbox("Select Visualization Type",

              ["Histogram", "Box Plot", "Scatter Plot", "Correlation Heatmap"])


  if viz_type == "Histogram":

    col = st.selectbox("Select column for histogram", data.select_dtypes(include=['int64',
'float64']).columns)

    bins = st.slider("Number of bins", 5, 100, 20)
```

```
        fig, ax = plt.subplots()
        ax.hist(data[col], bins=bins, edgecolor='black')
        ax.set_title(f"Histogram of {col}")
        ax.set_xlabel(col)
        ax.set_ylabel("Frequency")
        st.pyplot(fig)

    elif viz_type == "Box Plot":
        col = st.selectbox("Select column for box plot", data.select_dtypes(include=['int64',
'float64']).columns)
        fig, ax = plt.subplots()
        sns.boxplot(x=data[col], ax=ax)
        ax.set_title(f"Box Plot of {col}")
        st.pyplot(fig)

    elif viz_type == "Scatter Plot":
        col1 = st.selectbox("Select X-axis column", data.select_dtypes(include=['int64',
'float64']).columns)
        col2 = st.selectbox("Select Y-axis column", data.select_dtypes(include=['int64',
'float64']).columns)
        fig, ax = plt.subplots()
        ax.scatter(data[col1], data[col2], alpha=0.5)
        ax.set_title(f"Scatter Plot: {col1} vs {col2}")
        ax.set_xlabel(col1)
        ax.set_ylabel(col2)
        st.pyplot(fig)
```

```
    elif viz_type == "Correlation Heatmap":
        numeric_data = data.select_dtypes(include=['int64', 'float64'])
        fig, ax = plt.subplots(figsize=(10, 8))
        sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm', ax=ax)
        ax.set_title("Correlation Heatmap")
        st.pyplot(fig)

elif options == 'Logistic Regression Analysis(ML model)':
    # Logistic Regression Section
    st.header("3. Logistic Regression Analysis")

    # Create a binary target variable
    median_value = data['median_house_value'].median()
    data['high_value'] = (data['median_house_value'] > median_value).astype(int)

    # Feature Selection
    st.subheader("Feature Selection")
    available_features = data.select_dtypes(include=['int64', 'float64']).columns.tolist()
    columns_to_remove = ['median_house_value', 'high_value']
    selected_features = st.multiselect(
        "Select features for the model",
        [col for col in available_features if col not in columns_to_remove],
        default=['median_income', 'housing_median_age', 'total_rooms']
    )

    if len(selected_features) > 0:
```

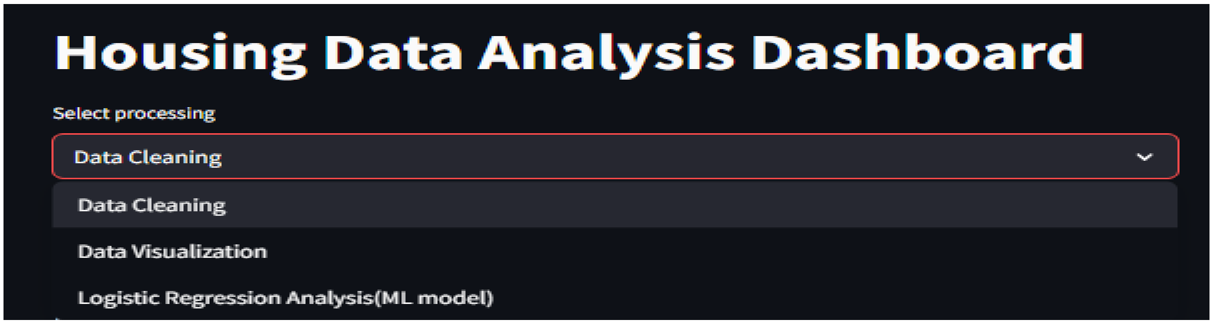
```
    # Prepare data
    X = data[selected_features]
    y = data['high_value']

    # Split data
    test_size = st.slider("Test set size", 0.1, 0.5, 0.2)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
random_state=42)

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train model
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_pred = model.predict(X_test_scaled)

    # Evaluation
    st.subheader("Model Evaluation")
    st.write("### Classification Report")
    report = classification_report(y_test, y_pred, output_dict=True)
    st.table(pd.DataFrame(report).transpose())
```

```
st.write("### Confusion Matrix")
cm = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots()
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
st.pyplot(fig)

# Coefficients
st.subheader("Model Coefficients")
coef_df = pd.DataFrame({
  'Feature': selected_features,
  'Coefficient': model.coef_[0]
})
st.table(coef_df.sort_values('Coefficient', ascending=False))

# Prediction on new data
st.subheader("Make Predictions")
st.write("Enter values for the selected features to predict if a house is high value:")
input_values = {}
col1, col2 = st.columns(2)
for i, feature in enumerate(selected_features):
  if i % 2 == 0:
    with col1:
```

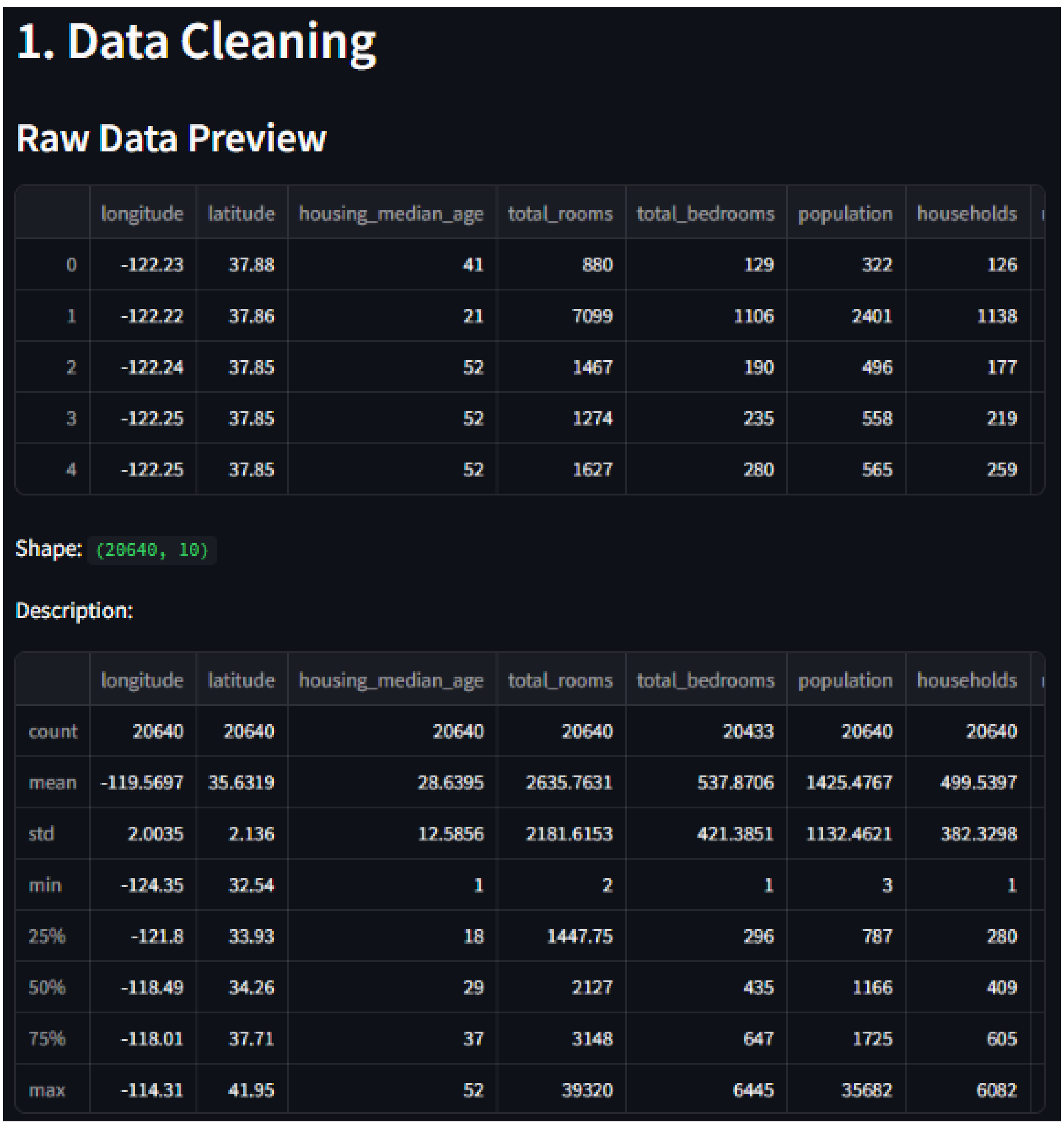
```
                input_values[feature] = st.number_input(feature,
value=float(data[feature].median()))
          else:
             with col2:
                input_values[feature] = st.number_input(feature,
value=float(data[feature].median()))


       if st.button("Predict"):
          input_df = pd.DataFrame([input_values])
          input_scaled = scaler.transform(input_df)
          prediction = model.predict(input_scaled)[0]
          probability = model.predict_proba(input_scaled)[0][1]
          st.write(f"### Prediction: {'High Value' if prediction == 1 else 'Not High Value'}")
          st.write(f"Probability of being high value: {probability:.2f}")
    else:
       st.warning("Please select at least one feature for the model.")
```

Streamlit app

Selection menu



Data cleaning



1. Data Cleaning

Raw Data Preview

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
0	-122.23	37.88	41	880	129	322	126
1	-122.22	37.86	21	7099	1106	2401	1138
2	-122.24	37.85	52	1467	190	496	177
3	-122.25	37.85	52	1274	235	558	219
4	-122.25	37.85	52	1627	280	565	259

Shape: `(20640, 10)`

Description:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
count	20640	20640	20640	20640	20433	20640	20640
mean	-119.5697	35.6319	28.6395	2635.7631	537.8706	1425.4767	499.5397
std	2.0035	2.136	12.5856	2181.6153	421.3851	1132.4621	382.3298
min	-124.35	32.54	1	2	1	3	1
25%	-121.8	33.93	18	1447.75	296	787	280
50%	-118.49	34.26	29	2127	435	1166	409
75%	-118.01	37.71	37	3148	647	1725	605
max	-114.31	41.95	52	39320	6445	35682	6082

Data Information

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #  Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0  longitude           20640 non-null  float64
 1  latitude            20640 non-null  float64
 2  housing_median_age  20640 non-null  float64
 3  total_rooms         20640 non-null  float64
 4  total_bedrooms      20433 non-null  float64
 5  population          20640 non-null  float64
 6  households          20640 non-null  float64
 7  median_income       20640 non-null  float64
 8  median_house_value  20640 non-null  float64
 9  ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

- [x] Encoding object columns

Label encoding applied.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #  Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0  longitude           20640 non-null  float64
 1  latitude            20640 non-null  float64
 2  housing_median_age  20640 non-null  float64
 3  total_rooms         20640 non-null  float64
 4  total_bedrooms      20433 non-null  float64
 5  population          20640 non-null  float64
 6  households          20640 non-null  float64
 7  median_income       20640 non-null  float64
 8  median_house_value  20640 non-null  float64
 9  ocean_proximity     20640 non-null  int32
dtypes: float64(9), int32(1)
memory usage: 1.5 MB
```

Duplicate Values

Number of duplicates: 0

- [ ] Handle duplicated values

Check Outliers (IQR method)

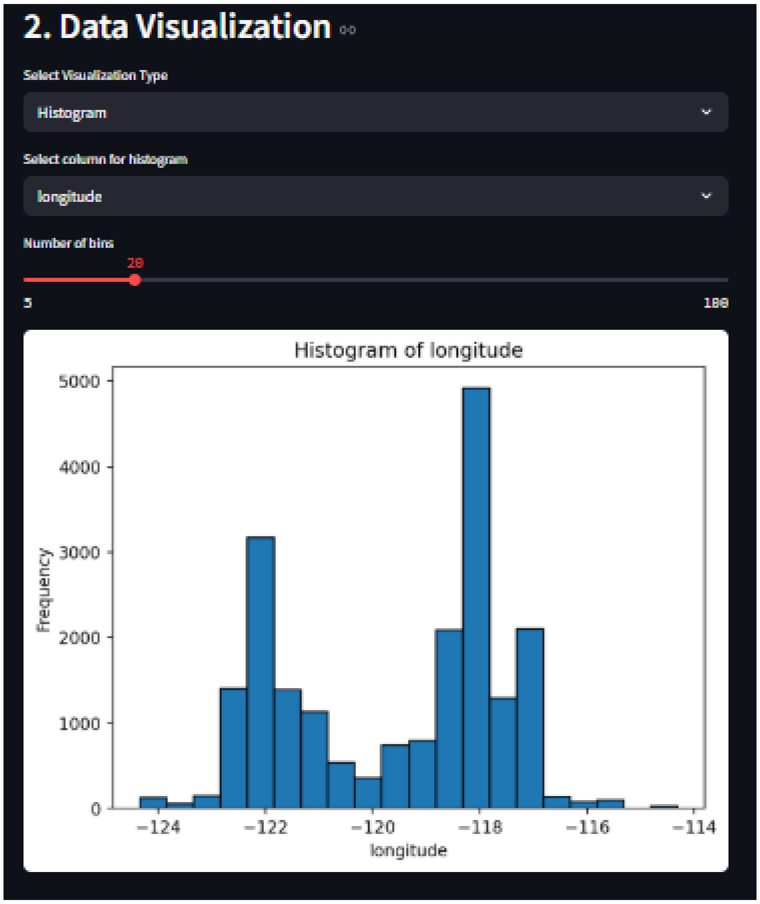
	Column	Outlier Count
0	longitude	0
1	latitude	0
2	housing_median_age	0
3	total_rooms	1287
4	total_bedrooms	1306
5	population	1196
6	households	1220
7	median_income	681
8	median_house_value	1071

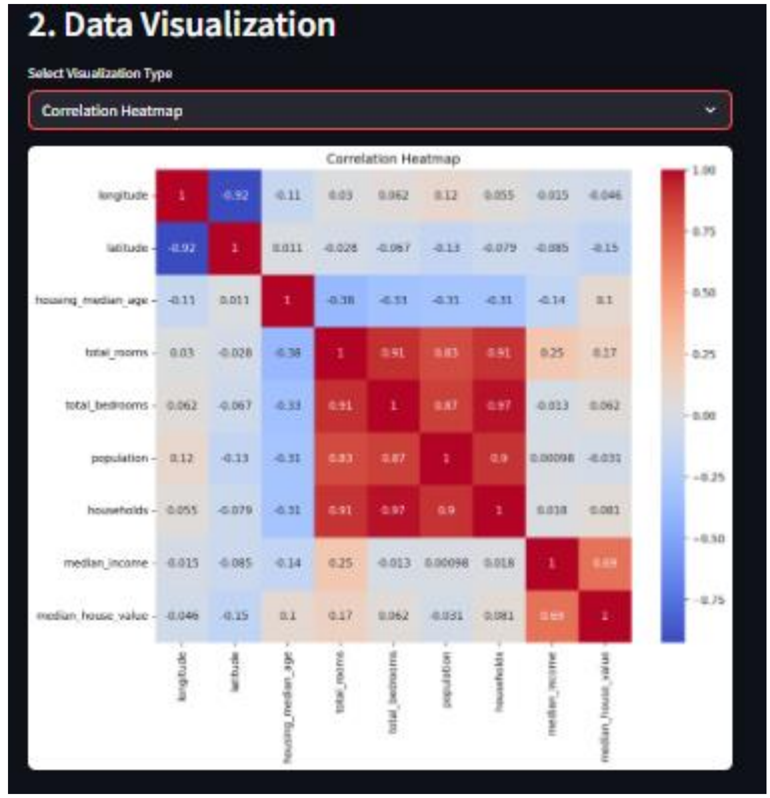
- [x] Handle outliers

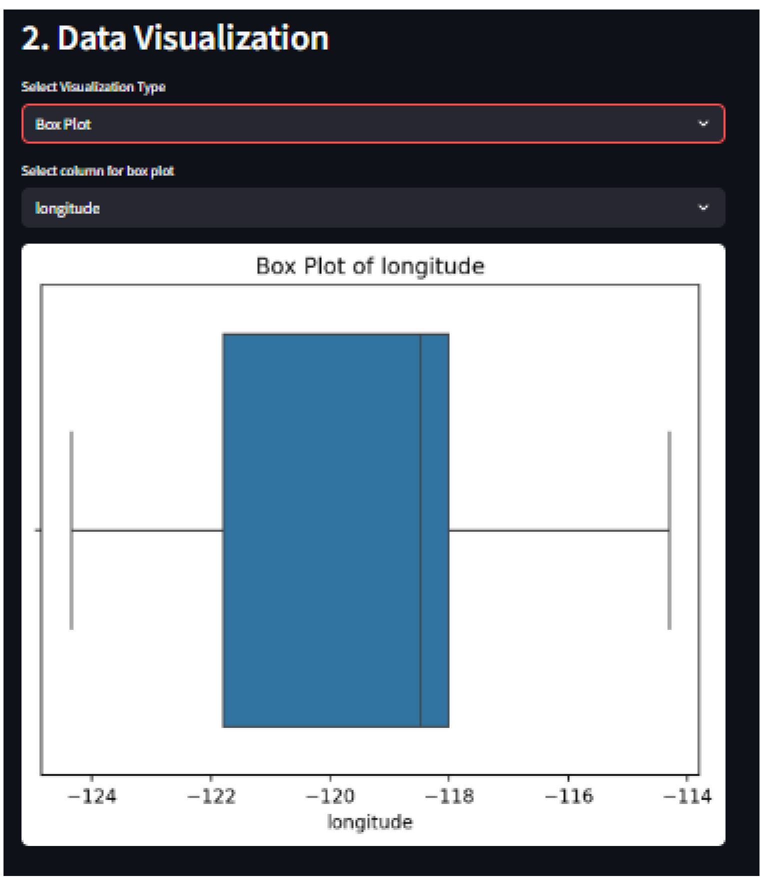
Outliers handled (capped to IQR limits)!

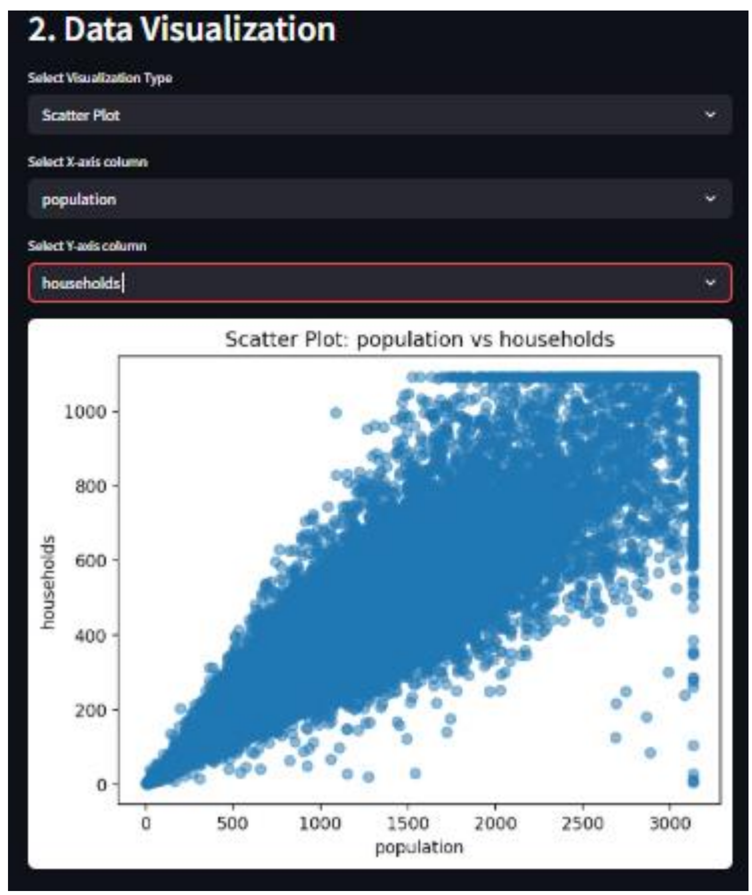
	Column	Outlier Count
0	longitude	0
1	latitude	0
2	housing_median_age	0
3	total_rooms	0
4	total_bedrooms	0
5	population	0
6	households	0
7	median_income	0
8	median_house_value	0

Sample of Data visualization









Model

3. Logistic Regression Analysis

Feature Selection

Select features for the model

median_income × housing_median... × total_rooms ×

longitude
latitude
total_bedrooms
population
households

Classification Report

	precision	recall	f1-score	support
0	0.7484	0.7963	0.7716	2,077.0000
1	0.7795	0.7289	0.7533	2,051.0000
accuracy	0.7628	0.7628	0.7628	0.7628
macro avg	0.7639	0.7626	0.7625	4,128.0000
weighted avg	0.7638	0.7628	0.7625	4,128.0000

Feature Selection

Select features for the model

median_income × housing_median... × total_rooms × population × longitude ×

Test set size

0.20

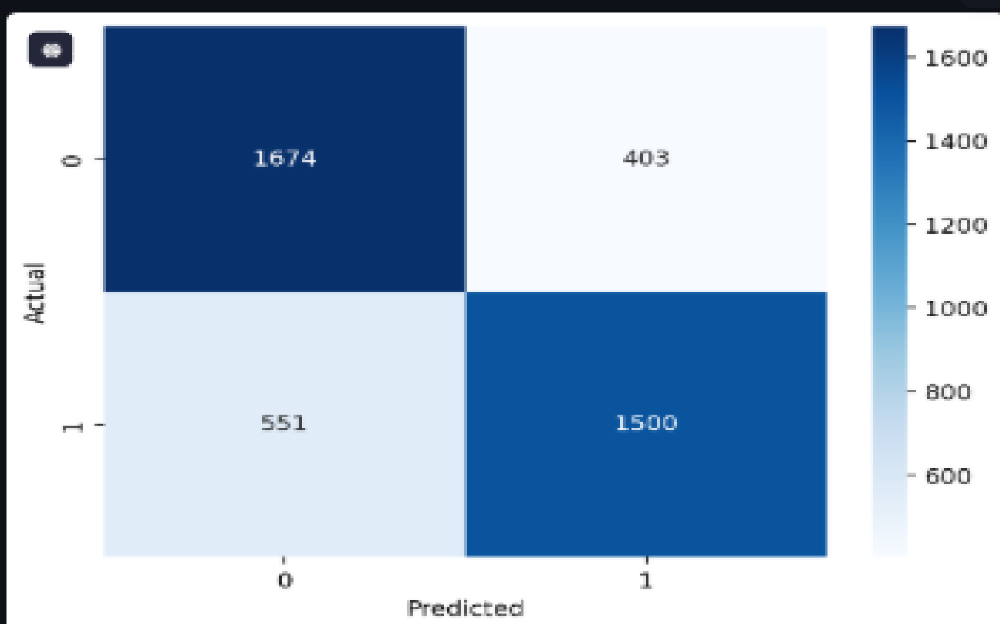
0.10 0.50

Model Evaluation

Classification Report

	precision	recall	f1-score	support
0	0.7524	0.8060	0.7782	2,077.0000
1	0.7882	0.7314	0.7587	2,051.0000
accuracy	0.7689	0.7689	0.7689	0.7689
macro avg	0.7703	0.7687	0.7685	4,128.0000
weighted avg	0.7702	0.7689	0.7685	4,128.0000

Confusion Matrix



Model Coefficients

	Feature	Coefficient
0	median_income	1.7952
1	housing_median_age	0.6100
2	total_rooms	0.4806
4	longitude	-0.0400
3	population	-0.2304

Prediction model