MACHINE LEARNING PROJECT

CONTENT

- Numeric Dataset
  - Data description
  - data cleaning and preprocessing
  - data analysis and exploration
  - machine model
- Images Dataset
  - Data description
  - data cleaning and preprocessing
  - data analysis and exploration
  - machine model

NUMERIC DATASET

DATA DESCRIPTION

This project focuses on predicting house prices using regression models (Linear Regression and K-Nearest Neighbors) based on real estate features. The goal is to analyze how attributes like square footage, number of bedrooms, location, and condition influence property prices.

The dataset provides a mix of structural, spatial, and quality features, making it ideal for price prediction. The project evaluates model performance to determine the most accurate approach for predicting house prices.

Source:https://www.kaggle.com/datasets/malakalaabiad/house-prices-regression

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	221900	3	1.00	1180	5650	1.0	N	0	Average	7	1180.0	0.0	1955	0	98178	47.5112	-122.257	1340.0	5650.0
1	538000	3	2.25	2570	7242	2.0	N	0	Average	7	2170.0	400.0	1951	1991	98125	47.7210	-122.319	1690.0	7639.0
2	180000	2	1.00	770	10000	1.0	N	0	Average	6	770.0	0.0	1933	0	98028	47.7379	-122.233	2720.0	8062.0
3	604000	4	3.00	1960	5000	1.0	N	0	Very Good	7	1050.0	910.0	1965	0	98136	47.5208	-122.393	1360.0	5000.0
4	510000	3	2.00	1680	8080	1.0	N	0	Average	8	1680.0	0.0	1987	0	98074	47.6168	-122.045	1800.0	7503.0

NUMERIC DATASET

importing libraries

```
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.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
```

DATA CLEANING AND PREPROCESSING

Overview of the data:

```
[ ] df.shape
```

```
(21613, 21)
```

```
df.columns
```

```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

```
[ ] #Check data types and nulls
    df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  int64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  object
 9   view           21613 non-null  int64
 10  condition      21613 non-null  object
 11  grade          21613 non-null  int64
 12  sqft_above     21611 non-null  float64
 13  sqft_basement  21612 non-null  float64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21612 non-null  float64
 20  sqft_lot15     21612 non-null  float64
dtypes: float64(8), int64(10), object(3)
memory usage: 3.5+ MB
```

Check for nulls and duplicates and drop the unnecessary columns:

```
#Null Values
df.isna().sum()
```

Show hidden output

```
[ ] df.dropna(inplace=True)
```

```
#Check Duplicated Data
df.duplicated().sum()
```

```
0
```

```
[ ] #drop id and date -Unnecessary columns-
    df.drop(columns=['id', 'date'], inplace=True)
    df.head()
```

DATA ANALYSIS AND EXPLORATON

numeric features of the data:

```
#Summary statistics
df.describe()
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	view	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
count	2.160800e+04	21608.000000	21608.000000	21608.000000	2.160800e+04	21608.000000	21608.000000	21608.000000	21608.000000	21608.000000	21608.000000	21608.000000	21608.000000	21608.000000	21608.000000	21608.000000	21608.000000
mean	5.401224e+05	3.370789	2.114830	2079.992734	1.510901e+04	1.494308	0.234358	7.656979	1788.445391	291.547344	1971.006387	84.421788	98077.939004	47.560063	-122.213883	1986.632682	12769.863569
std	3.671595e+05	0.930034	0.770168	918.503128	4.142508e+04	0.540009	0.766398	1.175558	828.132243	442.612836	29.369909	401.723661	53.505004	0.138560	0.140828	685.420089	27307.176542
min	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	1.000000	290.000000	0.000000	1900.000000	0.000000	98001.000000	47.155900	-122.519000	399.000000	651.000000
25%	3.218375e+05	3.000000	1.750000	1429.250000	5.040000e+03	1.000000	0.000000	7.000000	1190.000000	0.000000	1951.000000	0.000000	98033.000000	47.471075	-122.328000	1490.000000	5100.000000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.620000e+03	1.500000	0.000000	7.000000	1560.000000	0.000000	1975.000000	0.000000	98065.000000	47.571800	-122.230000	1840.000000	7620.000000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.069050e+04	2.000000	0.000000	8.000000	2210.000000	560.000000	1997.000000	0.000000	98118.000000	47.678000	-122.125000	2360.000000	10084.500000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	4.000000	13.000000	9410.000000	4820.000000	2015.000000	2015.000000	98199.000000	47.777600	-121.315000	6210.000000	871200.000000

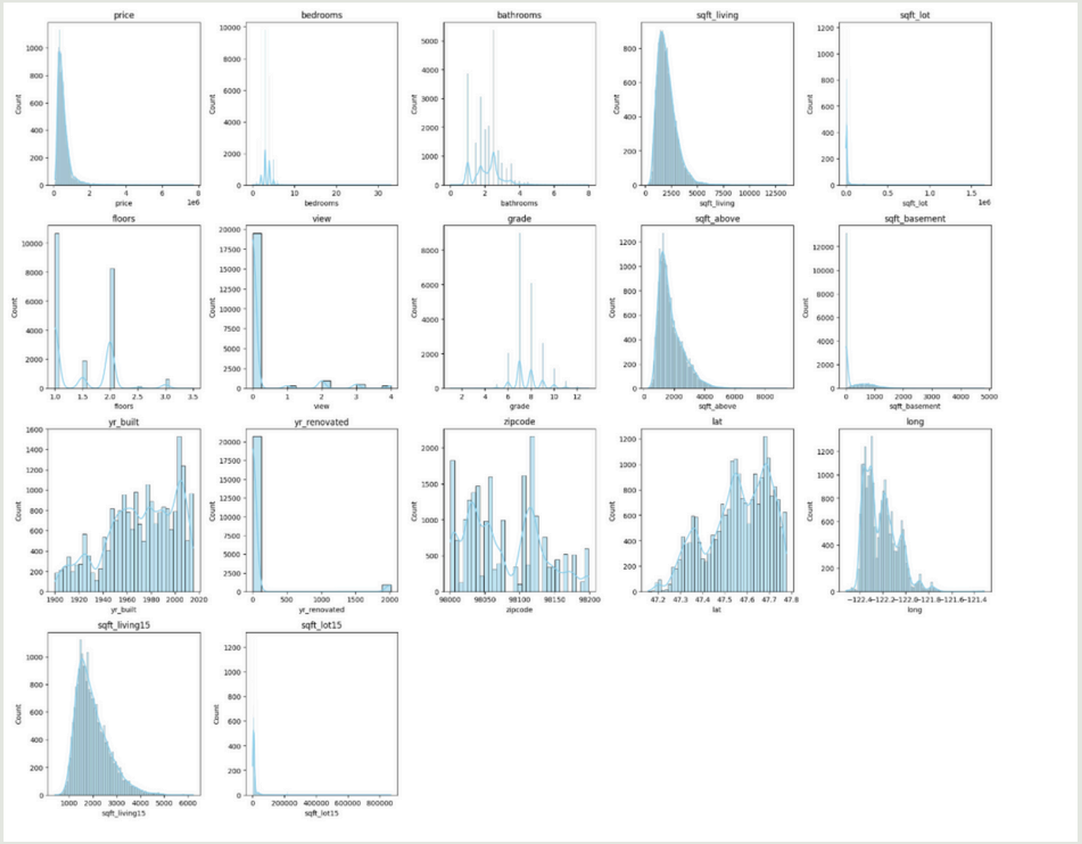
splitting the numeric and categorical data

```
num_columns = list(df.select_dtypes(include=np.number).columns)
num_columns
```

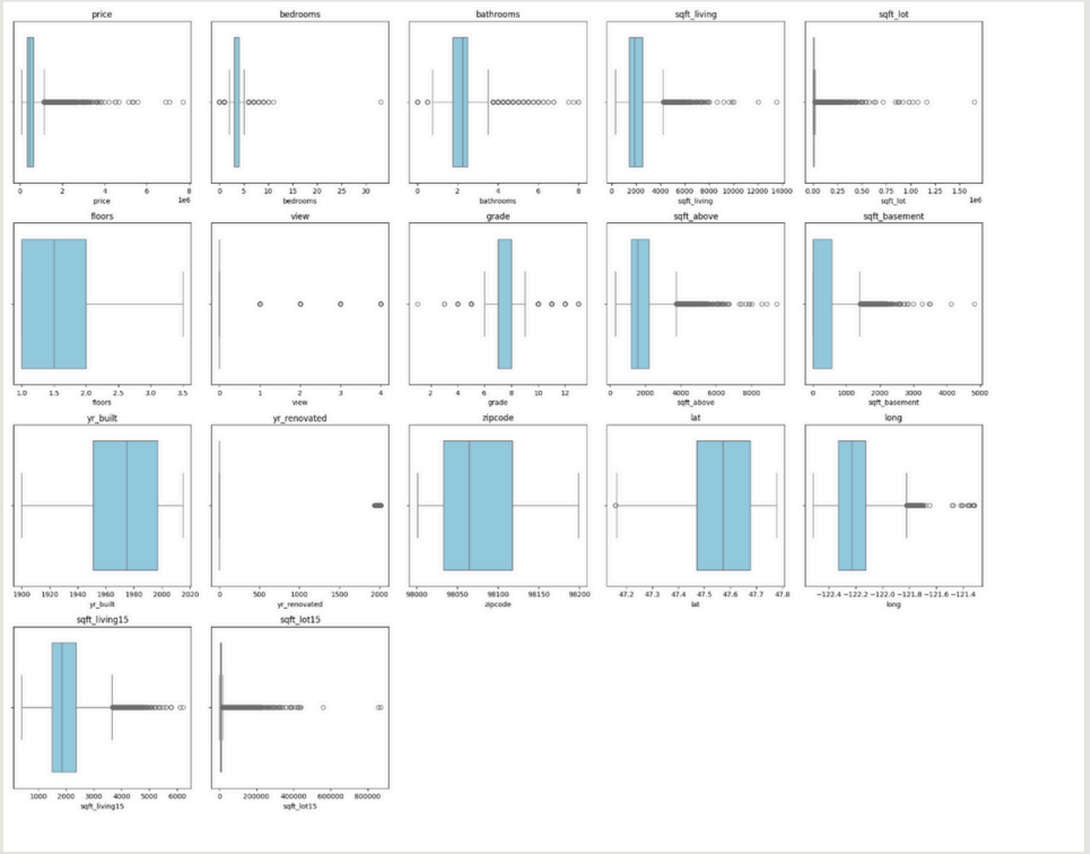
Show hidden output

```
[17] cat_columns = list(df.select_dtypes(include=object).columns)
     cat_columns
```

(freq) of numerical columns (histplot)



Boxplot for detecting outliers:



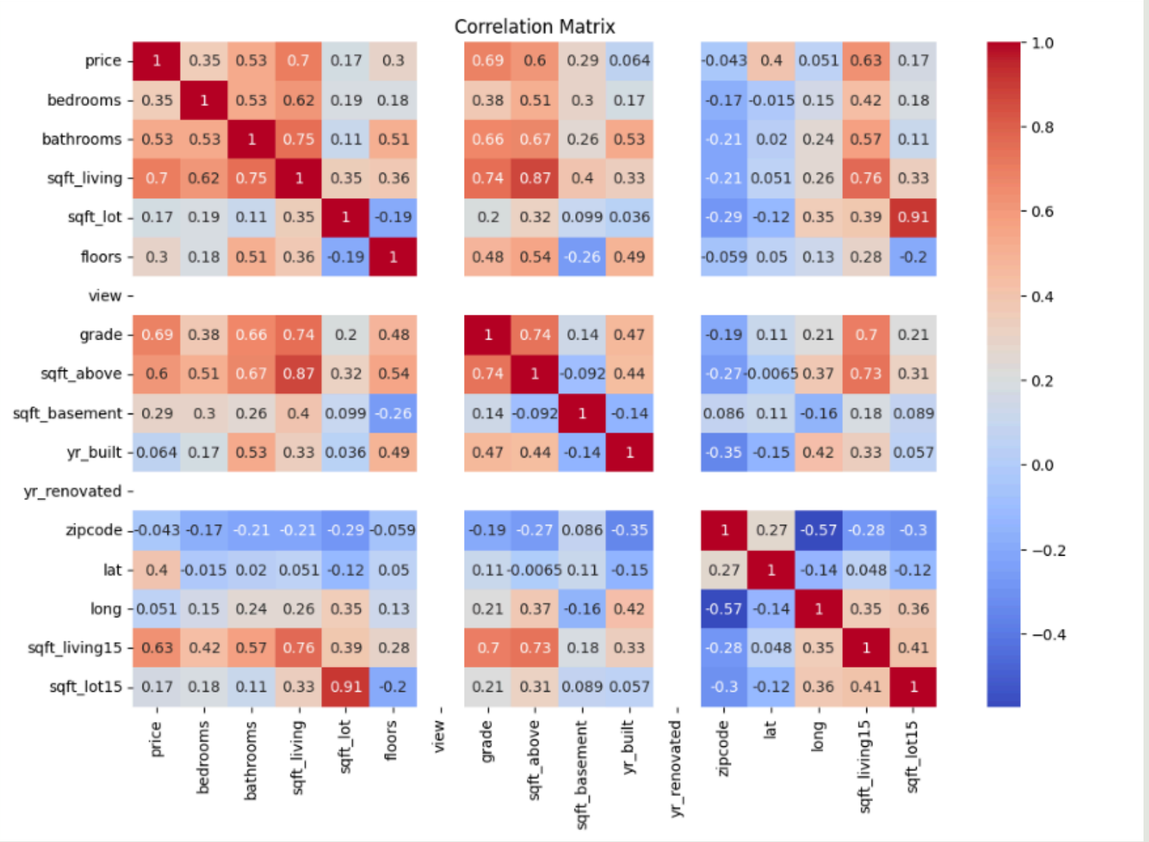
Detecting and handling outliers using Clip

```python
#detect outliers
for col in  num_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

    outliers= df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f"Column '{col}' has {len(outliers)} outliers.")
```

Show hidden output

```python
[25] def handle_outliers(df, column):
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)

    #  handle outliers usuing clip
    for col in num_columns:
        handle_outliers(df, col)
```

Correlation matrix(heatmap)



Encoding and Normalization

```
[ ]  #Encoding
     from sklearn.preprocessing import LabelEncoder
     le = LabelEncoder()
     for column in cat_columns:
         df[column] = le.fit_transform(df[column])
```

```
[ ]  #Normalization

     from sklearn.preprocessing import MinMaxScaler
     scaler = MinMaxScaler()
     df[num_columns] = scaler.fit_transform(df[num_columns])
```

THE FINAL RESULT OF THE DATA

```
df.head()
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	lat	long	sqft_living15	sqft_lot15
0	0.139276	0.375	0.125000	0.225824	0.275122	0.0	0	0.0	0	0.375	0.257971	0.000000	0.478261	0.0	0.893939	0.568172	0.375089	0.288120	0.295620
1	0.438969	0.375	0.541667	0.578515	0.360501	0.4	0	0.0	0	0.375	0.544928	0.285714	0.443478	0.0	0.626263	0.908253	0.286328	0.395285	0.413240
2	0.099550	0.125	0.125000	0.121793	0.508413	0.0	0	0.0	0	0.125	0.139130	0.000000	0.286957	0.0	0.136364	0.935647	0.409449	0.710655	0.438255
3	0.501544	0.625	0.791667	0.423737	0.240263	0.0	0	0.0	4	0.375	0.220290	0.650000	0.565217	0.0	0.681818	0.583734	0.180387	0.294244	0.257181
4	0.412422	0.375	0.458333	0.352691	0.405443	0.0	0	0.0	0	0.625	0.402899	0.000000	0.756522	0.0	0.368687	0.739347	0.678597	0.428965	0.405198

MACHINE LEARNING MODELS

Libraries Import and Data splitting

```
[36] from sklearn.linear_model import LinearRegression
     from sklearn.neighbors import KNeighborsRegressor
     from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
[37] x = df.drop(columns=['price'])
     y = df['price']

     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(17286, 18)
(4322, 18)
(17286,)
(4322,)
```

LINEAR REGRESSION MODEL

Linear Regression Model

```
[ ] LR_model = LinearRegression()
    LR_model.fit(x_train, y_train)
```

```
LinearRegression()
```

```
[39] LR_pred = LR_model.predict(x_test)
     LR_pred
```

```
array([0.4490888 , 0.41545384, 0.49379774, ..., 0.77757231, 0.44796291,
       0.11045249])
```

LINEAR REGRESSION MODEL

checking how accurate is the model

```
[35] mse = mean_squared_error(y_test, y_pred)
     mae = mean_absolute_error(y_test, y_pred)
     r2 = r2_score(y_test, y_pred)
     print("Mean Squared Error:", mse)
     print("Mean Absolute Error:", mae)
     print("R-squared:", r2)
```

```
Mean Squared Error: 0.014433378793033252
Mean Absolute Error: 0.09169565427767065
R-squared: 0.746373905154708
```

KNN MODEL

KNN Model

```
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(x_train, y_train)
```

```
KNeighborsRegressor
KNeighborsRegressor()
```

```
[44] KNN_Pred = knn_model.predict(x_test)
     KNN_Pred
```

```
array([0.46277496, 0.31040715, 0.3884261 , ..., 0.9719849 , 0.43337541,
       0.1583323 ])
```

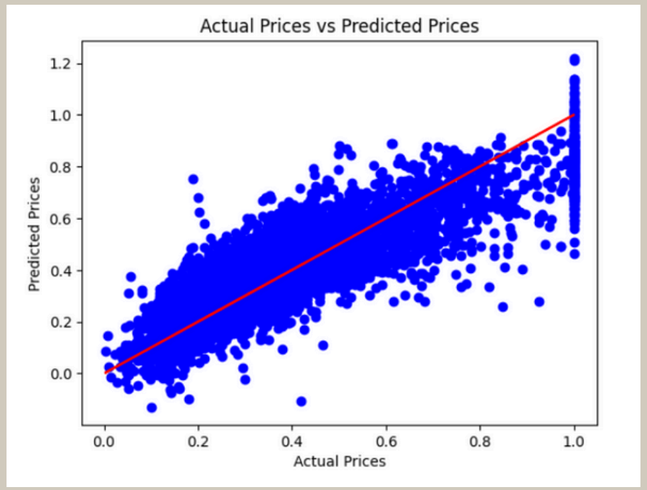
KNN MODEL

checking how accurate is the model

```
[45] mse = mean_squared_error(y_test, KNN_Pred)
     mae = mean_absolute_error(y_test, KNN_Pred)
     r2 = r2_score(y_test, KNN_Pred)
     print("Mean Squared Error:", mse)
     print("Mean Absolute Error:", mae)
     print("R-squared:", r2)
```

```
Mean Squared Error: 0.009663816664011206
Mean Absolute Error: 0.06733237422908768
R-squared: 0.8301855638281264
```

Comparison betweent the two models




```
Mean Squared Error: 0.014433378793033252
Mean Absolute Error: 0.09169565427767065
R-squared: 0.746373905154708
```


```
Mean Squared Error: 0.009663816664011206
Mean Absolute Error: 0.06733237422908768
R-squared: 0.8301855638281264
```

The KNN Regression model outperforms the Linear Regression model across all evaluation metrics, making it the better choice for predicting house prices in this dataset. KNN demonstrates greater accuracy and better fits the data.

IMAGE DATASET

DATA DESCRIPTION

The dataset used is from the 'Facial Expression Recognition Challenge'. It includes grayscale images (48x48 pixels) with associated emotion labels. The emotion classes are: Angry, Disgust, Fear, Happy, and Sad.
The dataset was preprocessed to remove Classes 5 and 6 (Neutral and Surprise).

MACHINE LEARNING MODELS

LOGISTIC REGRESSION

IMAGE PREPROCESSING

-Images were normalized by dividing pixel values by 255

.- HOG (Histogram of Oriented Gradients) features were extracted to represent each image.

```
# Extract HOG Features
def extract_hog_features(images):
    hog_features = []
    for image in images:
        features, _ = feature.hog(image.reshape(48, 48), visualize=True, block_norm='L2-Hys')
        hog_features.append(features)
    return np.array(hog_features)

X_train_hog = extract_hog_features(X_train)
X_test_hog = extract_hog_features(X_test)
```

train the model

```
# Train a Logistic Regression Model
model = LogisticRegression(max_iter=1000, random_state=42)
losses = []
for epoch in range(1, 11):  # Simulating 10 training epochs
    model.fit(X_train_hog, y_train)
    y_prob = model.predict_proba(X_train_hog)
    loss = log_loss(y_train, y_prob)
    losses.append(loss)
    print(f"Epoch {epoch}: Loss = {loss:.4f}")
```

KNN MODEL

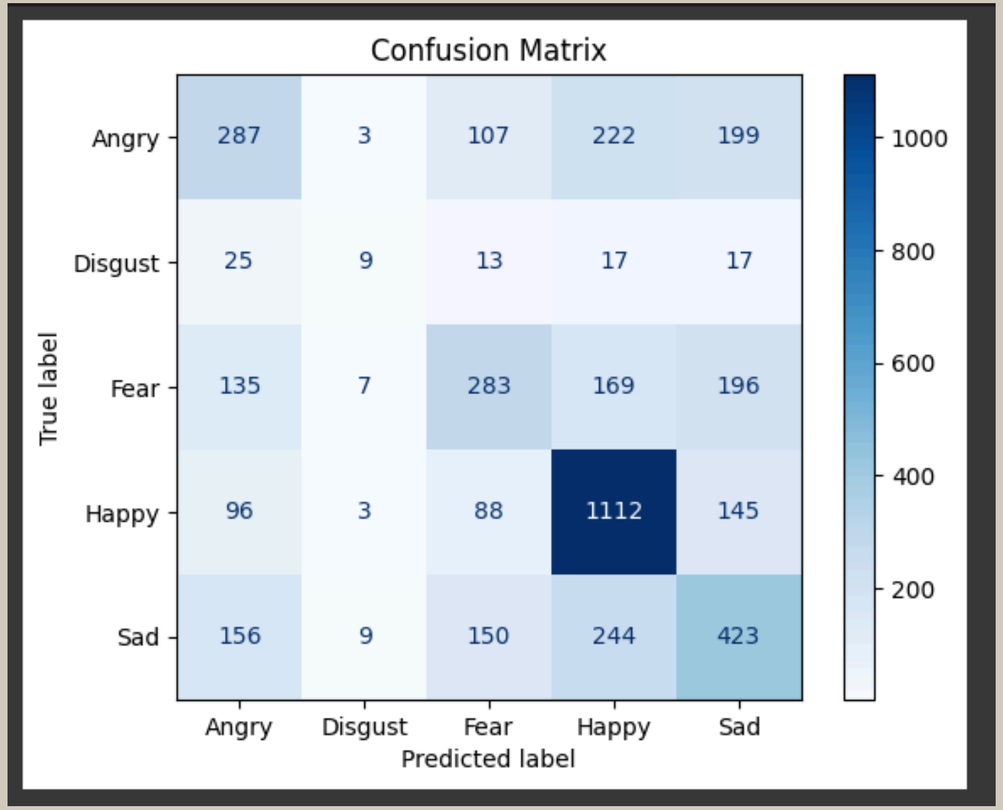
IMAGE PREPROCESSING

- PCA (Principal Component Analysis) features were extracted to represent each image.

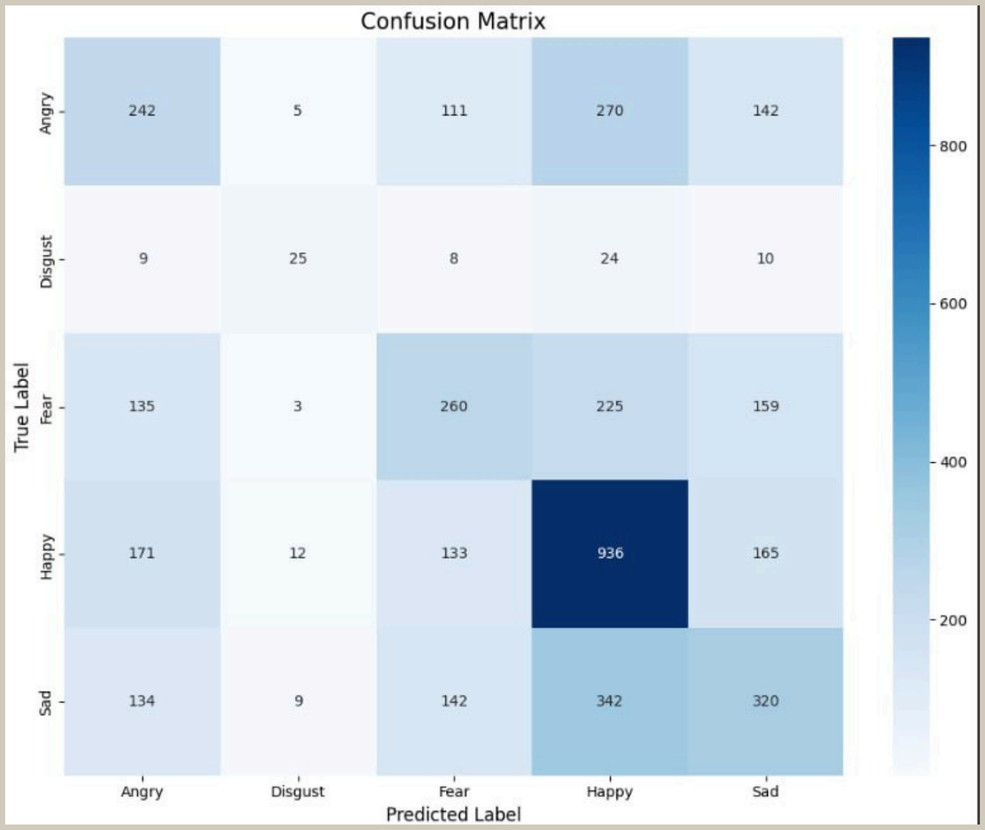
```
# PCA with more components for better variance capture
pca = PCA(n_components=n_components, random_state=42)
X_pca = pca.fit_transform(X_normalized)

# Stratified train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y, test_size=test_size, stratify=y, random_state=42
)

# Advanced KNN with weighted voting
knn = KNeighborsClassifier(n_neighbors=7, weights='distance')
knn.fit(X_train, y_train)

# Predictions
y_pred = knn.predict(X_test)
y_pred_proba = knn.predict_proba(X_test)
```

Comparison betweent the two models

Logistic Reg.



KNN



Comparison betweent the two models

	Logistic Reg.	KNN
ACCURACY	0.51373	0.4466
LOSS	1.0638	5.0473
RECALL	0.513	0.394

Thank You