**Ex___3**

name:saksham kaushish.....rollno:23126045

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

name:saksham kaushish.....rollno:23126045

```
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
print(df.head())
```

```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude
0    -122.23
1    -122.22
2    -122.24
3    -122.25
4    -122.25
```

name:saksham kaushish.....rollno:23126045

```
X=data.data
y = data.target

print("The features are", X[:5])
print("Target variable", y[:5])
```

```
The features are [[ 8.32520000e+00  4.10000000e+01  6.98412698e+00  1.02380952e+00
   3.22000000e+02  2.55555556e+00  3.78800000e+01 -1.22230000e+02]
 [ 8.30140000e+00  2.10000000e+01  6.23813708e+00  9.71880492e-01
   2.40100000e+03  2.10984183e+00  3.78600000e+01 -1.22220000e+02]
 [ 7.25740000e+00  5.20000000e+01  8.28813559e+00  1.07344633e+00
   4.96000000e+02  2.80225989e+00  3.78500000e+01 -1.22240000e+02]
 [ 5.64310000e+00  5.20000000e+01  5.81735160e+00  1.07305936e+00
   5.58000000e+02  2.54794521e+00  3.78500000e+01 -1.22250000e+02]
 [ 3.84620000e+00  5.20000000e+01  6.28185328e+00  1.08108108e+00
   5.65000000e+02  2.18146718e+00  3.78500000e+01 -1.22250000e+02]]
Target variable [4.526 3.585 3.521 3.413 3.422]
```

Double-click (or enter) to edit

```
x_shape=np.shape(X)
y_shape=np.shape(y)
print(x_shape)
print(y_shape)
```

```
(20640, 8)
(20640,)
```

name:saksham kaushish.....rollno:23126045

```
def custom_train_test_split(X, y, test_size=0.2):


   X_train=X[:int(np.shape(X)[0]*(1-test_size))]
   X_test=X[int(np.shape(X)[0]*(1-test_size)):]
   y_train=y[:int(np.shape(X)[0]*(1-test_size))]
   y_test=y[int(np.shape(X)[0]*(1-test_size)):]
   return X_train, X_test, y_train, y_test
```

```
X_train, X_test, y_train, y_test=custom_train_test_split(X,y,test_size=0.2)
print(np.shape(X_train))
print(np.shape(X_test))
```

```
(16512, 8)
(4128, 8)
```

```
print(np.shape(X_train))
print(np.shape(X_test))
print(np.shape(y_train))
print(np.shape(y_test))
```

```
(16512, 8)
(4128, 8)
(16512, 8)
(4128, 8)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(np.shape(X_train))
print(np.shape(X_test))
```

```
(16512, 8)
(4128, 8)
```

```
print(np.shape(X_train))
print(np.shape(X_test))
print(np.shape(y_train))
print(np.shape(y_test))
```

```
(16512, 8)
(4128, 8)
(16512,)
(4128,)
```

name:saksham kaushish.....rollno:23126045

```
def custom_standard_scaler(X):
    X_scaled = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
    mean = np.mean(X, axis=0)
    std = np.std(X, axis=0)
    return X_scaled, mean, std
```

```
X_train_scaled, train_mean, train_std = custom_standard_scaler(X_train)
print(X_train_scaled[:5])
```

```
[[-0.326196    0.34849025 -0.17491646 -0.20836543  0.76827628  0.05137609
  -1.3728112   1.27258656]
 [-0.03584338  1.61811813 -0.40283542 -0.12853018 -0.09890135 -0.11736222
  -0.87669601  0.70916212]
 [ 0.14470145 -1.95271028  0.08821601 -0.25753771 -0.44981806 -0.03227969
  -0.46014647 -0.44760309]
 [-1.01786438  0.58654547 -0.60001532 -0.14515634 -0.00743434  0.07750687
  -1.38217186  1.23269811]
 [-0.17148831  1.14200767  0.3490073   0.08662432 -0.48587717 -0.06883176
   0.5320839  -0.10855122]]
```

name:saksham kaushish.....rollno:23126045

```
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the data (for training set)
X_train_scaled = scaler.fit_transform(X_train)
print(X_train_scaled[:5])
```

```
[[ 2.41761472  0.91662721  0.59834018 -0.15167507 -0.96121719 -0.08812743
   1.25839198 -1.61169516]
 [ 2.40489816 -0.64722457  0.31754578 -0.25284611  0.83005604 -0.17496218
   1.24880693 -1.60649422]
 [ 1.84707938  1.77674569  1.08917568 -0.05496988 -0.81129822 -0.04006405
   1.2440144  -1.6168961 ]
 [ 0.98454409  1.77674569  0.15915997 -0.05572379 -0.75787882 -0.08961009
   1.2440144  -1.62209704]
 [ 0.02444394  1.77674569  0.33400078 -0.04009543 -0.75184759 -0.161008
   1.2440144  -1.62209704]]
```

name:saksham kaushish.....rollno:23126045

```
import numpy as np
```

```
M = X_train_scaled.shape[0]  # Number of rows
print(M)
# Add a column of ones as the first column using vstack
ones_column = np.ones((M,1))
print(np.shape(ones_column))
print(np.shape(X_train_scaled))
X_train_scaled = np.hstack((ones_column, X_train_scaled))

print(np.shape(X_train_scaled))
```

```
16512
(16512, 1)
(16512, 8)
(16512, 9)
```

```
# theta_0=np.zeros(9)
# print(theta_0)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
# h=np.dot(X,theta_0)
```

```
# num_iters = 1500
# alpha = 0.01
```

name:saksham kaushish.....rollno:23126045

name: saksham Kaushish .... rollno.: 23126045

```
import numpy as np

def compute_cost(X, y, M, theta):
    J=np.sum((X.dot(theta)-y)**2)/(2*M)
    return J

def gradient_descent(X, y, M, theta_0, alpha, num_iters):
    """Python version of gradientDescent.m."""
    J_history = np.zeros(num_iters)
    theta = theta_0.copy()
    for i in range(num_iters):
       #To be completed by student
       J_history[i] = compute_cost(X, y, M, theta)
       theta = theta - (alpha/M)*(X.T.dot(X.dot(theta)-y))

    return theta, J_history
```

name: saksham Kaushish .... rollno.: 23126045

```
theta = np.zeros((X_train_scaled.shape[1],1))
print(np.shape(theta))
```

```
alpha = 0.01  # Learning rate
iterations = 1000
M = X_train_scaled.shape[0]
y_train=y_train.reshape(-1,1)
print(np.shape(y_train))
# Run gradient descent
theta_optimized, cost_history = gradient_descent(X_train_scaled, y_train, M, theta, alpha, ite
```

```
(9, 1)
(16512, 1)
```

name: saksham Kaushish .... rollno.: 23126045

```
print("Optimized parameters:")
print(theta_optimized)
```

```
Optimized parameters:
[[ 2.07185749]
 [ 0.82894365]
 [ 0.17853146]
 [-0.13794939]
 [ 0.15669182]
 [ 0.01681517]
 [-0.04522857]
 [-0.48705563]
 [-0.45147126]]
```

```
X_test_scaled = (X_test - train_mean) / train_std
X_test_scaled = np.hstack((np.ones((X_test_scaled.shape[0], 1)), X_test_scaled))
m=np.shape(X_test_scaled)[0]
```

name: saksham Kaushish .... rollno.: 23126045

```
def evaluate_model(X, y, theta):
    """Evaluate model using test data"""
    predictions = X.dot(theta)
    X = np.array(X) # Ensure numpy array
    y = np.array(y)  # Ensure numpy array
    y = y.reshape(-1)
    print(np.shape(y))
    predictions.flatten()
    print(np.shape(predictions))
    #To be completed by student

    mse=np.mean(((predictions-y)/2)**2)
    rmse=np.sqrt(mse)

    return mse, rmse

# Make predictions and evaluate
test_mse, test_rmse = evaluate_model(X_test_scaled, y_test, theta_optimized)

print("\nModel Performance:")
print(f"MSE: {test_mse:.4f}")
print(f"RMSE: {test_rmse:.4f}")
```

```
(4128,)
(4128, 1)

Model Performance:
MSE: 0.5075
RMSE: 0.7124
```

name: saksham Kaushish .... rollno.: 23126045

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

def evaluate_regression_model(y_true, y_pred):
    """
    Compute evaluation metrics for regression models.

    Parameters:
    - y_true: array-like of shape (n_samples,) - True target values.
    - y_pred: array-like of shape (n_samples,) - Predicted target values.

    Returns:
    - dict: A dictionary containing MAE, MSE, RMSE, and R².
    """
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    y_true = y_true.reshape(-1)  # Reshape to 1D
    y_pred = y_pred.reshape(-1)

    # Mean Absolute Error (MAE)
    mae = mean_absolute_error(y_true, y_pred)

    # Mean Squared Error (MSE)
    mse = mean_squared_error(y_true, y_pred)
```

```
    # Root Mean Squared Error (RMSE)
    rmse = np.sqrt(mse)

    # R-squared (R²)
    r2 = r2_score(y_true, y_pred)

    # Return a dictionary with all metrics
    metrics = {
        "Mean Absolute Error (MAE)": mae,
        "Mean Squared Error (MSE)": mse,
        "Root Mean Squared Error (RMSE)": rmse,
        "R-squared (R²)": r2
    }

    return metrics
```

Start coding or generate with AI.

name: saksham Kaushish .... rollno.: 23126045

```
y_predict = X_test_scaled.dot(theta_optimized)
metrics = evaluate_regression_model(y_test, y_predict)

# Print the evaluation metrics
for metric, value in metrics.items():
    print(f"{metric}: {value}")
```

```
Mean Absolute Error (MAE): 0.5476758462432642
Mean Squared Error (MSE): 0.5671852986082032
Root Mean Squared Error (RMSE): 0.7531170550506762
R-squared (R²): 0.5671692517174325
```

name: saksham Kaushish .... rollno.: 23126045

```
test_size=0.2
y_train=y[:int(np.shape(X)[0]*(1-test_size))]
y_test=y[int(np.shape(X)[0]*(1-test_size)):]
min1=min(y_test)
max1=max(y_test)
min2=min(y_train)
max2=max(y_train)

def max_min(x,y):
  for(i,j) in zip(x,y):
    print((i-min1)/(max1-min1),(j-min2)/(max2-min2))
```

name: saksham Kaushish .... rollno.: 23126045

```
from sklearn.preprocessing import MinMaxScaler

# create data
data = np.column_stack((X_train, y_train))
```

```
# scale features
scaler = MinMaxScaler()
model=scaler.fit(data)
scaled_data=model.transform(data)

# print scaled features
print(scaled_data)
```

```
[[0.19032151 0.62745098 0.02927784 ... 0.01702128 0.72908367 0.90226638]
 [0.22845202 0.94117647 0.02541945 ... 0.12978723 0.61653386 0.70824656]
 [0.25216204 0.05882353 0.03373236 ... 0.22446809 0.38545817 0.69505074]
 ...
 [0.16789424 0.68627451 0.02196727 ... 0.15744681 0.59462151 0.3651552 ]
 [0.35994676 0.2745098  0.03904731 ... 0.53510638 0.23804781 0.25567111]
 [0.14314285 1.         0.01782502 ... 0.55531915 0.19223108 0.34515528]]
```

name: saksham Kaushish .... rollno.: 23126045

```
from sklearn.preprocessing import RobustScaler
x= np.column_stack((X_train, y_train))
transformer = RobustScaler().fit(X)
transformer
RobustScaler()
transformer.transform(X)
```

```
array([[ 2.1975824 ,  0.63157895,  1.08893505, ..., -0.30798124,
         0.95767196, -0.98680739],
       [ 2.18666422, -0.42105263,  0.62606588, ..., -0.83080046,
         0.95238095, -0.98416887],
       [ 1.70773218,  1.21052632,  1.89804168, ..., -0.01859871,
         0.94973545, -0.98944591],
       ...,
       [-0.84170929, -0.63157895, -0.0146346 , ..., -0.57767639,
         1.36772487, -0.72031662],
       [-0.76500677, -0.57894737,  0.06228597, ..., -0.81512066,
         1.36772487, -0.74670185],
       [-0.525816  , -0.68421053,  0.01587687, ..., -0.23592945,
         1.35185185, -0.72559367]])
```

```
from google.colab import drive
drive.mount('/content/drive')
path="/content/drive/MyDrive/IML_lab"
```

```
already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/cont
```

name: saksham Kaushish .... rollno.: 23126045

```
import pandas as pd
from sklearn.feature_selection import mutual_info_regression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
# Load dataset
df = pd.read_csv("/content/drive/MyDrive/IML_lab/house-prices-advanced-regression-techniques.z

# Drop non-informative columns
df = df.drop(columns=["Id"])

# Separate target variable
y = df["SalePrice"]
X = df.drop(columns=["SalePrice"])

# Encode categorical variables
X_encoded = X.copy()
for col in X.select_dtypes(include=["object"]).columns:
    X_encoded[col] = LabelEncoder().fit_transform(X[col].astype(str))

# Handle missing values by filling with median
X_encoded = X_encoded.fillna(X_encoded.median())

# Apply Mutual Information for feature selection
mi_scores = mutual_info_regression(X_encoded, y)
mi_scores_series = pd.Series(mi_scores, index=X_encoded.columns).sort_values(ascending=False)
top_5_features = mi_scores_series.head(5).index.tolist()

# Select only the top 5 features
X_selected = X_encoded[top_5_features]

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state

# Train Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)


y_pred_test = model.predict(X_test)


mae_test = mean_absolute_error(y_test, y_pred_test)
mse_test = mean_squared_error(y_test, y_pred_test)
rmse_test = mse_test ** 0.5

# Print results
print("Mean Absolute Error (MAE) on Test Set:", mae_test)
print("Mean Squared Error (MSE) on Test Set:", mse_test)
print("Root Mean Squared Error (RMSE) on Test Set:", rmse_test)
```

```
Mean Absolute Error (MAE) on Test Set: 25517.055650295166
Mean Squared Error (MSE) on Test Set: 1603752877.6968725
Root Mean Squared Error (RMSE) on Test Set: 40046.88349543411
```

name: saksham Kaushish .... rollno.: 23126045

```
import pandas as pd
from sklearn.feature_selection import mutual_info_regression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load dataset
df = pd.read_csv("/content/drive/MyDrive/IML_lab/house-prices-advanced-regression-techniques

# Drop non-informative columns
df = df.drop(columns=["Id"])

# Separate target variable
y = df["SalePrice"]
X = df.drop(columns=["SalePrice"])

# Encode categorical variables
X_encoded = X.copy()
for col in X.select_dtypes(include=["object"]).columns:
    X_encoded[col] = LabelEncoder().fit_transform(X[col].astype(str))

# Handle missing values by filling with median
X_encoded = X_encoded.fillna(X_encoded.median())

# Apply Mutual Information for feature selection
mi_scores = mutual_info_regression(X_encoded, y)
mi_scores_series = pd.Series(mi_scores, index=X_encoded.columns).sort_values(ascending=False
top_5_features = mi_scores_series.head(5).index.tolist()

# Select only the top 5 features
X_selected = X_encoded[top_5_features]

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_sta

# Train Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate error metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5

# Print results
print("Top 5 Features:", top_5_features)
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
```

```
Top 5 Features: ['OverallQual', 'Neighborhood', 'GrLivArea', 'GarageCars', 'TotalBsmtSF']
Mean Absolute Error (MAE): 25517.055650295166
Mean Squared Error (MSE): 1603752877.6968725
Root Mean Squared Error (RMSE): 40046.88349543411
```

**learnings** Feature scaling significantly impacts model performance and convergence. Gradient