I N D E X

Name Shuvom Dhar

Class _____ Section 1 Roll No. 1043 Year 3rd

Subject Machine Learning Lab

Sl. No.	Experiment Description	Experiment Date	Submission Date	Remarks Signature
1.1	Fuel Consumption			
1.2	Used Car Price Prediction			
2	Housing Price Prediction			
3	Salary Prediction			
4	China GDP			
5	Cancer Prediction			

1. Perform linear regression to predict
a) CO_2 Emission
b) The selling price of a used car.
Evaluate the quality of the models by computing relevant performance metrics, including the R^2 value. Generate and display a plot that compares the actual values to the predicted values (Actual vs Predicted) for both tasks.
Dataset : fuel_consumption_dataset.csv
Dataset : used_cars_dataset.csv

1.a)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_square_error

#Load the dataset
df = pd.read_csv('fuel_consumption_dataset.csv')

#Display first few rows
print(df.head())
```

No. 1.1

```
# Select relevant features and target variable
features = ['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTI
           ON_COMB']
target = 'CO2EMISSIONS'

X = df[features]
y = df[target]

# Split the data into training and testing sets (80% train,
  20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state = 42)

# Initialize and train the linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict CO2 emissions
y_pred = model.predict(X_test)

# Evaluate model performance
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f"R² Score : {r2:.4f}")
```

Teacher's Signature .....

No.
1.1

```
print(f"Mean Squared Error: {mse:.4f}")

# Plot Actual vs Predicted values
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(),
          y_test.max()], color='red', linestyle='dashed')
plt.xlabel("Actual CO2 Emissions")
plt.ylabel("Predicted CO2 Emissions")
plt.title("Actual vs Predicted CO2 Emissions")
plt.show()
```

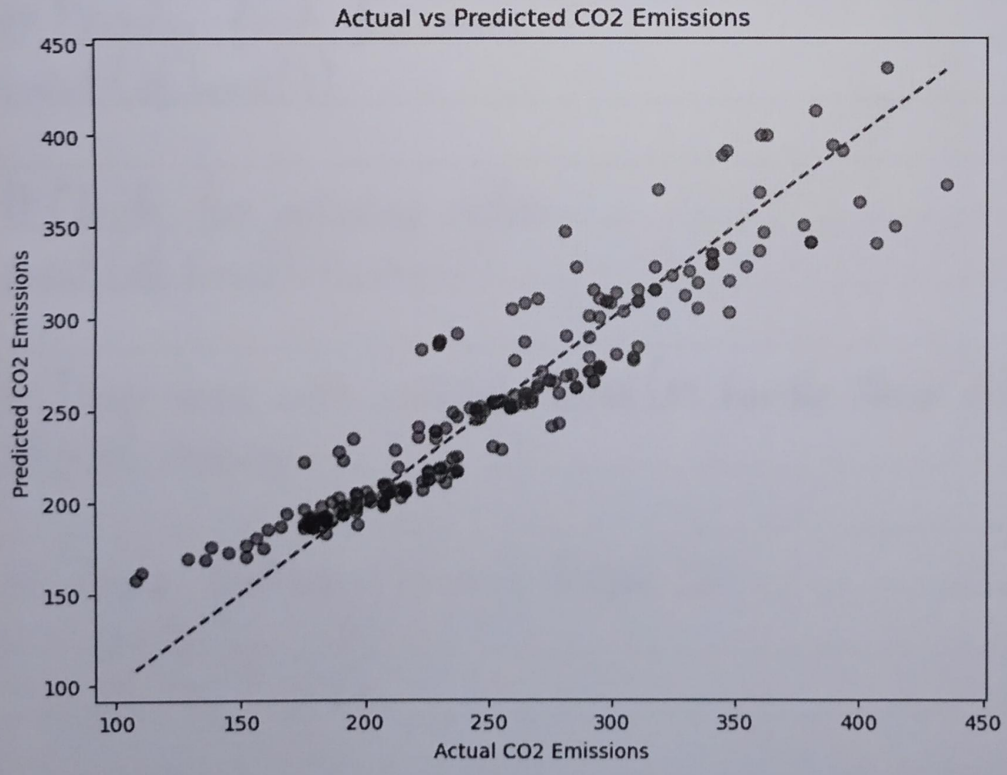
	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	\
0	2014	ACURA	ILX	COMPACT	2.0	4	
1	2014	ACURA	ILX	COMPACT	2.4	4	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	

	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	\
0	AS5	Z	9.9	6.7	
1	M6	Z	11.2	7.7	
2	AV7	Z	6.0	5.8	
3	AS6	Z	12.7	9.1	
4	AS6	Z	12.1	8.7	

	FUELCONSUMPTION_COMB	FUELCONSUMPTION_COMB_MPG	CO2EMISSIONS
0	8.5	33	196
1	9.6	29	221
2	5.9	48	136
3	11.1	25	255
4	10.6	27	244

R^2 Score: 0.8760

Mean Squared Error: 512.8551



1. b)

```
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 LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

# Load dataset
df = pd.read_csv('used cars_dataset.csv')

# Display first few rows
print(df.head())

# Check for missing values
print(df.isnull().sum())

# Drop rows with missing values (or handle them differently)
df = df.dropna()

# Choose features (X) and target (y)
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

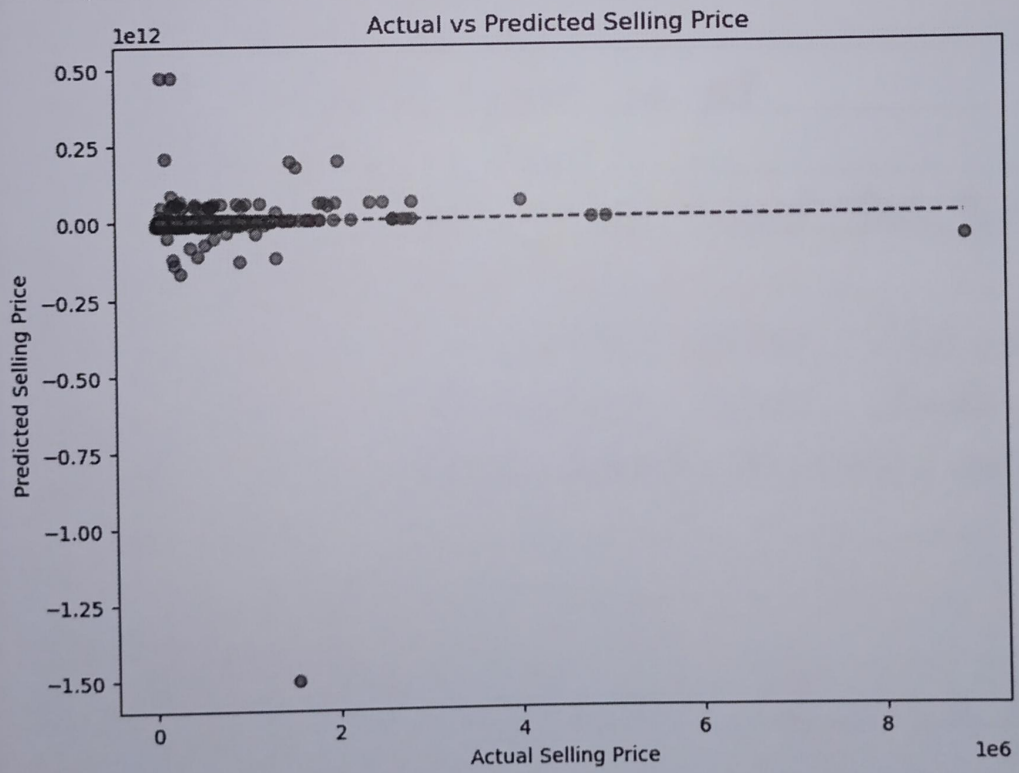
1.2

```
# Handle categorical data if present
X = pd.get_dummies(X, drop_first = True)

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict selling price
y_pred = model.predict(X_test)

# Evaluate model performance
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f"R² Score: {r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")

# Plot Actual vs Predicted prices
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='blue', alpha = 0.5)
```

Teacher's Signature .....

```
plt.plot([y_test.min(), y_test.max()], [y_test.min(),
          y_test.max()], color='red', linestyle='dashed')
plt.xlabel("Actual Selling Price")
plt.ylabel("Predicted Selling Price")
plt.title("Actual vs Predicted Selling Price")
plt.show()
```

```
                      name  year  km_driven    fuel seller_type transmission  \
0             Maruti 800 AC  2007      70000  Petrol  Individual       Manual
1  Maruti Wagon R LXI Minor  2007      50000  Petrol  Individual       Manual
2      Hyundai Verna 1.6 SX  2012     100000  Diesel  Individual       Manual
3    Datsun RediGO T Option  2017      46000  Petrol  Individual       Manual
4     Honda Amaze VX i-DTEC  2014     141000  Diesel  Individual       Manual

          owner  selling_price
0   First Owner          60000
1   First Owner         135000
2   First Owner         600000
3   First Owner         250000
4  Second Owner         450000
name             0
year             0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
selling_price    0
dtype: int64
R² Score: -11689013742.7553
Mean Squared Error: 35671396682280987525120.0000
```



Expt. No. 2

Page No. 7
Date. 16/3/2025

2) Perform linear regression with L1 (Lasso) and L2 (Ridge) regularization to predict the price of a House. Use hyper-parameter tuning for the best result. Evaluate the accuracy of the models by computing relevant performance metrics, including the R^2 value. Generate and display a plot that compares the actual values to the predicted values (Actual vs Predicted) for both tasks.

Dataset : housing_price_dataset.csv

```
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, GridSearchCV
from sklearn.linear_model import Ridge, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error

# Load dataset
df = pd.read_csv('housing_price_dataset.csv')

# Display first few rows
```

```
print(df.head())

# Check for non-numeric columns
print(df.dtypes)

# Drop columns that are clearly non-numeric
df = df.drop(columns=['Address'], errors='ignore')

# Handle missing values (drop or impute)
df = df.dropna()

# Convert categorical variables into numerical using One-Hot Encoding
df = pd.get_dummies(df, drop_first=True)

# Ensure target variable is numeric
df.iloc[:,-1] = pd.to_numeric(df.iloc[:,-1], errors='coerce')

# Drop any remaining NaN values (if any exist after conversion)
df = df.dropna()

# Select features (X) and target variable (y)
X = df.iloc[:, :-1]
y = df.iloc[:,-1]
```

```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
y, test_size = 0.2, random_state = 42)

# Hyperparameter tuning
param_grid = {'alpha': np.logspace(-3, 3, 7)}

# Lasso Regression (L1)
lasso = Lasso(max_iter = 10000)
lasso_grid = GridSearchCV(lasso, param_grid, scoring = 'r2',
                          cv = 5)
lasso_grid.fit(X_train, y_train)
best_lasso = lasso_grid.best_estimator_

# Ridge Regression (L2)
ridge = Ridge()
ridge_grid = GridSearchCV(ridge, param_grid, scoring = 'r2',
                          cv = 5)
ridge_grid.fit(X_train, y_train)
best_ridge = ridge_grid.best_estimator_
```

Expt. No. 2

```
# Predictions
y_pred_lasso = best_lasso.predict(X_test)
y_pred_ridge = best_ridge.predict(X_test)

# Performance Metrics
r2_lasso = r2_score(y_test, y_pred_lasso)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)

r2_ridge = r2_score(y_test, y_pred_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)

print(f"Lasso Regression - Best Alpha: {lasso_grid.best_params_
    ['alpha']}")
print(f"Lasso Regression - R² Score: {r2_lasso:.4f}, MSE:
    {mse_lasso:.4f}")

print(f"Ridge Regression - Best Alpha: {ridge_grid.best_params_
    ['alpha']}")
print(f"Ridge Regression - R² Score: {r2_ridge:.4f}, MSE:
    {mse_ridge:.4f}")

# Plot Actual vs Predicted Prices
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
```

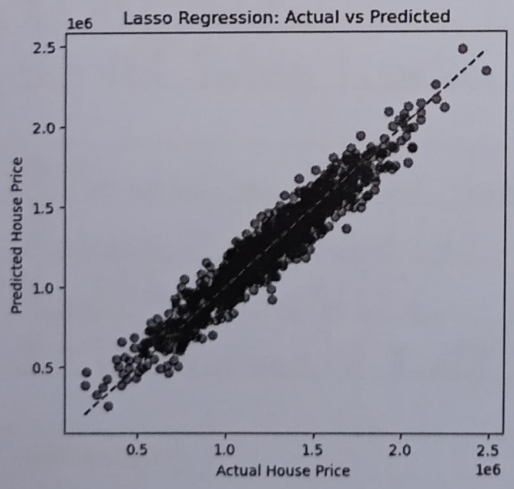
```
plt.scatter(y_test, y_pred_ridge, color='green', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max
    ()], color='red', linestyle='dashed')
plt.xlabel("Actual House Price")
plt.ylabel("Predicted House Price")
plt.title("Ridge Regression: Actual vs Predicted")
plt.show()
```

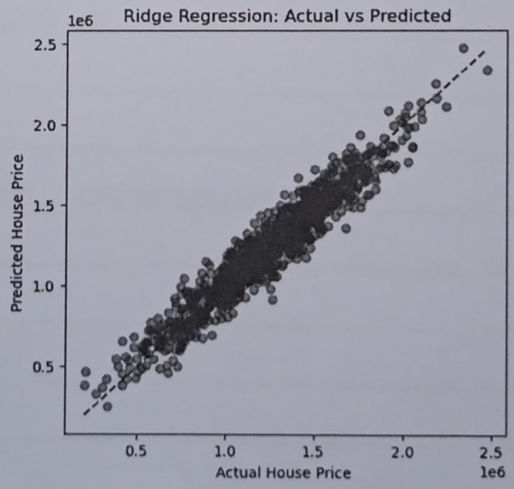
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	\
0	79545.458574	5.682861	7.009188	
1	79248.642455	6.002900	6.730821	
2	61287.067179	5.865890	8.512727	
3	63345.240046	7.188236	5.586729	
4	59982.197226	5.040555	7.839388	

	Avg. Area Number of Bedrooms	Area Population	Price	\
0	4.09	23086.800503	1.059034e+06	
1	3.09	40173.072174	1.505891e+06	
2	5.13	36882.159400	1.058988e+06	
3	3.26	34310.242831	1.260617e+06	
4	4.23	26354.109472	6.309435e+05	

	Address
0	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	USS Barnett\nFPO AP 44820
4	USNS Raymond\nFPO AE 09386

```
Avg. Area Income                float64
Avg. Area House Age             float64
Avg. Area Number of Rooms       float64
Avg. Area Number of Bedrooms    float64
Area Population                 float64
Price                           float64
Address                          object
dtype: object
Lasso Regression - Best Alpha: 0.1
Lasso Regression - R² Score: 0.9180, MSE: 10089007626.9052
Ridge Regression - Best Alpha: 1.0
Ridge Regression - R² Score: 0.9180, MSE: 10089005431.7701
```





DATE 17/3/25 | PAGE NO. 12 | EXPT. NO. 3

3) Perform linear regression with one feature using gradient descent (without using library function) to predict the salary of an employee based on the feature YearsExperience. Use hyper-parameter tuning for the best result. Plot the hypothesis function and the data points after each epoch. Evaluate the accuracy of the models by computing relevant performance metrics, including the R^2 value.
Dataset: salary_dataset.csv

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

# Load dataset
df = pd.read_csv('salary_data.csv')

# Extract feature (YearsExperience) and target (Salary)
X = df['YearsExperience'].values
y = df['Salary'].values

# Normalize X for faster convergence
X_mean = np.mean(X)
X_std = np.std(X)
X = (X - X_mean) / X_std
```

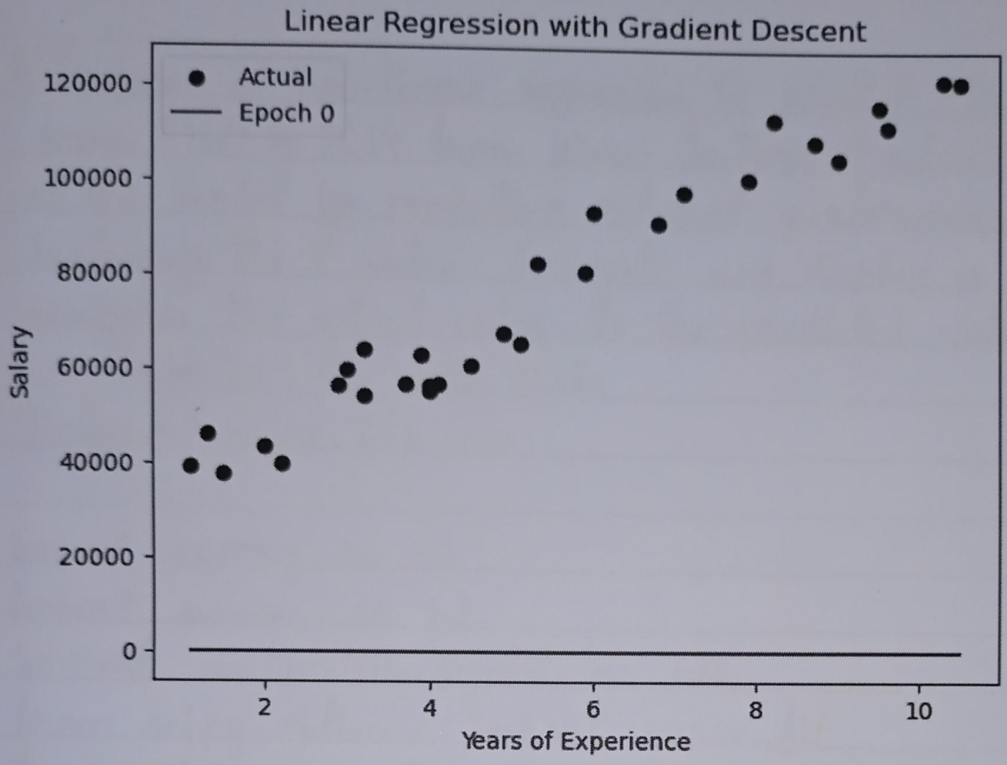
DATE 17/3/25 | PAGE NO. 13 | EXPT. NO. 3

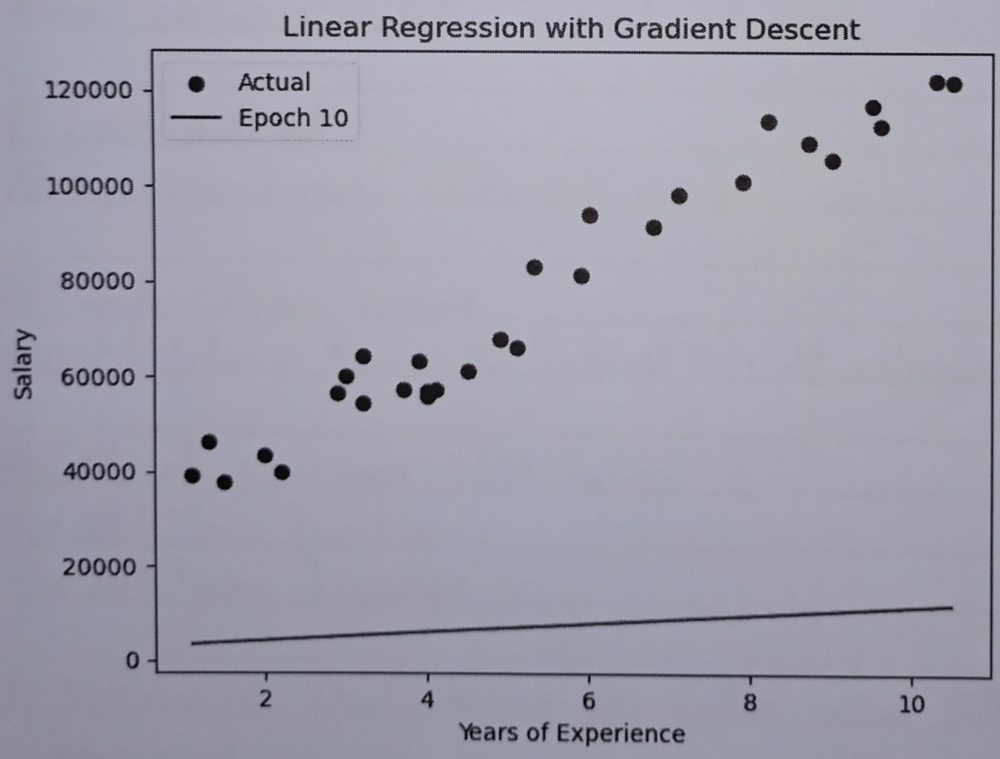
```
# Add bias term (X0 = 1)
X = np.c_[np.ones(X.shape[0]), X]

# Initialize parameters (theta0, theta1)
theta = np.zeros(2)

# Hyperparameters
learning_rate = 0.01
epochs = 100

# Number of training examples
m = len(y)

# Cost function history
cost_history = []

# Gradient Descent
for epoch in range(epochs):
    # Compute predictions
    y_pred = X.dot(theta)

    # Compute error
    error = y_pred - y

    # Compute gradients
    gradients = (1/m) * X.T.dot(error)
```

DATE 17/3/25 | PAGE NO. 14 | EXPT. NO. 3

```
#Update parameters
theta -= learning_rate * gradients

#Compute cost (Mean Squared Error)
cost = (1/(2*m)) * np.sum(error**2)
cost_history.append(cost)

#Plot regression line every 10 epochs
if epoch % 10 == 0:
    plt.scatter(df['YearsExperience'], y, color='blue',
                label='Actual')
    plt.plot(df['YearsExperience'], y_pred, color='red',
             label=f'Epoch {epoch}')
    plt.xlabel('Years of Experience')
    plt.ylabel('Salary')
    plt.title('Linear Regression with Gradient Descent')
    plt.legend()
    plt.show()

#Final model predictions
y_pred_final = X.dot(theta)

#Compute R^2 Score
r2 = r2_score(y, y_pred_final)

print(f"Final Parameters: Theta0 = {theta[0]:.4f}, Theta1 =
```

```
{theta[1]:.4f}")
print(f"Final R² Score: {r2:.4f}")

#Plot Cost Function Convergence
plt.plot(range(epochs), cost_history, color='green')
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Cost Function Convergence')
plt.show()
```


Linear Regression with Gradient Descent
Actual
Epoch 0
120000
100000
80000
60000
40000
20000
0
Salary
2
4
6
8
10
Years of Experience


Linear Regression with Gradient Descent
Actual
Epoch 10
120000
100000
80000
60000
40000
20000
0
Salary
2
4
6
8
10
Years of Experience

DATE 17/3/25 | PAGE NO. 16 | EXPT. NO. 4

4) Perform a non-linear regression to predict China's GDP from 1960 to 2014 from given features. Evaluate the quality of the model by computing relevant performance metrics, including the R^2 value. Generate and display a plot that compares the actual values to the predicted values (Actual vs Predicted) for both tasks.
Dataset: china_gdp.csv

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

# Load dataset
df = pd.read_csv('china_gdp.csv')

#Check column names
print("Column Names in Dataset:", df.columns)

# Extract year and GDP values
X = df['Year'].values
y = df['Value'].values

# Normalize GDP values for better curve fitting
y_max = np.max(y)
```

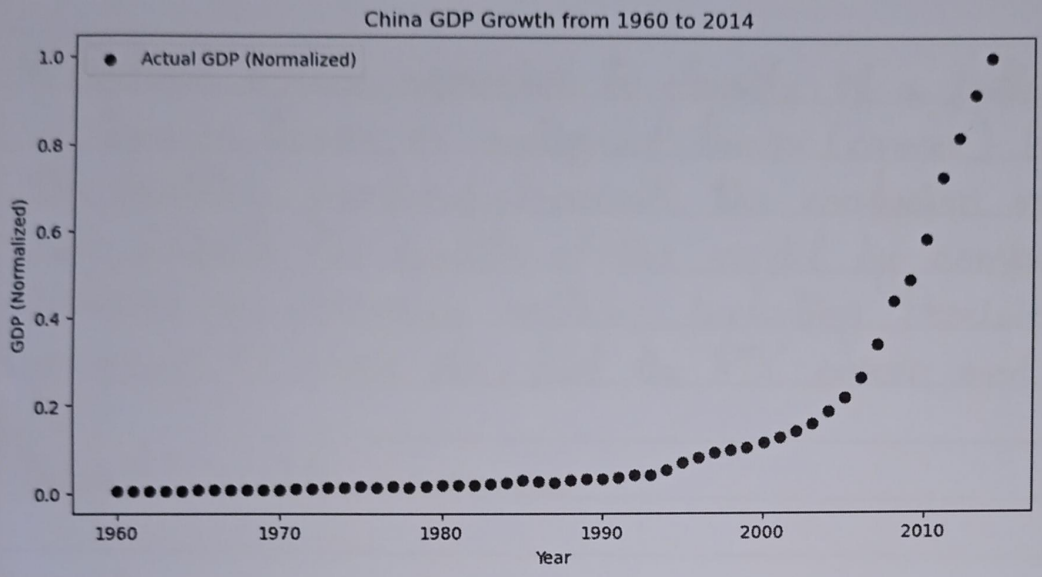
DATE 18/3/25 | PAGE NO. 17 | EXPT. NO. 4

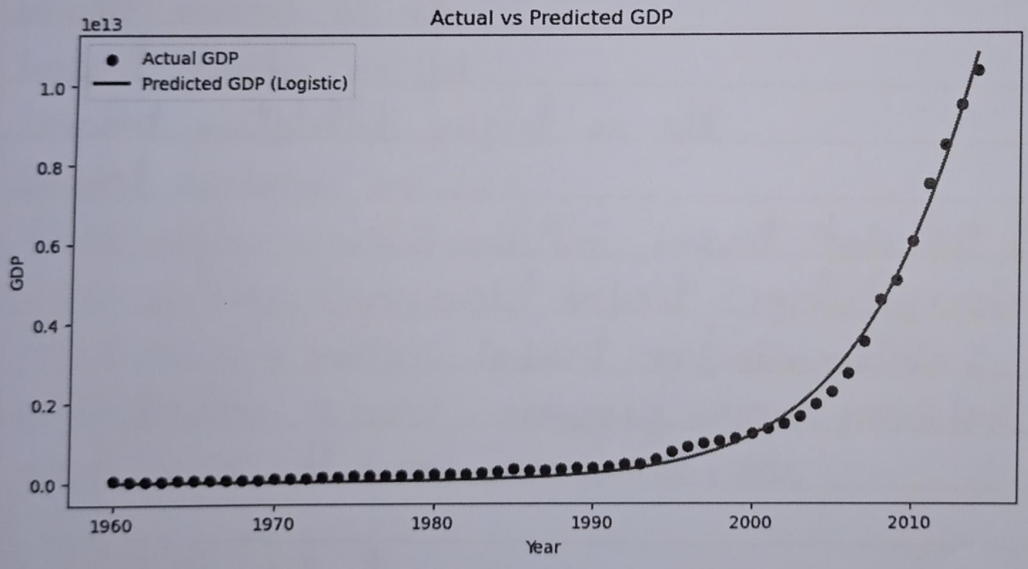
```
y = y / y_max

# Visualizing the dataset
plt.figure(figsize=(10,5))
plt.scatter(X, y, color='blue', label='Actual GDP
            (Normalized)')
plt.xlabel('Year')
plt.ylabel('GDP (Normalized)')
plt.title('China GDP Growth from 1960 to 2014')
plt.legend()
plt.show()

# Define Logistic Growth Function
def logistic_function(x, a, b, c):
    return c/(1+np.exp(-a*(x-b)))

# Provide initial parameter guesses to help curve_fit()
initial_guesses=[0.1, 2000, 1]

# Curve Fitting using scipy.optimize.curve_fit()
popt, _ = curve_fit(logistic_function, X, y, p0=initial_
                    guesses, maxfev=50000)

# Extract optimized parameters
a_opt, b_opt, c_opt = popt
```

DATE 18/3/25 | PAGE NO. 18 | EXPT. NO. 4

```
# Generate predictions
y_pred = logistic_function(X, a_opt, b_opt, c_opt)

# Denormalize predictions
y_pred = y_pred * y_max

# Compute R^2 Score
r2 = r2_score(df['Value'], y_pred)

#Plot Actual vs Predicted GDP
plt.figure(figsize=(10,5))
plt.scatter(df['Year'], df['Value'], color='blue', label=
            'Actual GDP')
plt.plot(df['Year'], y_pred, color='red', linewidth=2,
         label='Predicted GDP (Logistic)')
plt.xlabel('Year')
plt.ylabel('GDP')
plt.title('Actual vs Predicted GDP')
plt.legend()
plt.show()

print(f"Optimized Parameters: a={a_opt:.4f}, b={b_opt=.4
      f}, c={c_opt:.4f}")
print(f"Final R^2 Score: {r2:.4f}")
```

```
Column Names in Dataset: Index(['Year', 'Value'], dtype='object')
```






```
Optimized Parameters: a=0.1752, b=2021.3700, c=4.8272
Final R² Score: 0.9938
```

DATE 18/3/25 | PAGE NO. 19 | EXPT. NO. 5

5) Perform logistic regression to classify if a patient has a benign tumor or malignant tumor (cancer) based on the features provided. Generate the confusion matrix and evaluate the quality of the model by computing relevant performance metrics including Precision, Recall, accuracy, F1-Score etc. Plot the ROC curve and calculate AUC.

Dataset : samples_cancer.csv

```
import numpy as np
import pandas as pd
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 confusion_matrix,
classification_report, accuracy_score, precision_score,
recall_score, f1_score, roc_curve, auc

# Load the dataset
df = pd.read_csv('samples_cancer.csv')

# Display dataset info
print("Dataset Info:")
print(df.info())
```

DATE 18/3/25 | PAGE NO. 20 | EXPT. NO. 5

```
# Drop 'ID' column as it's not relevant for classification
df.drop(columns=['ID'], inplace=True)

# Convert 'BareNuc' column to numeric (if it contains '?' replace with NaN)
df['BareNuc'] = pd.to_numeric(df['BareNuc'], errors='coerce')

# Fill missing values with median of the column
df.fillna(df.median(), inplace=True)

# Define features and target variable
X = df.drop(columns=['Class'])
y = df['Class'].map({2:0, 4:1})

# Split data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                   test_size=0.2, random_
                                   state=42)

# Train Logistic Regression Model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

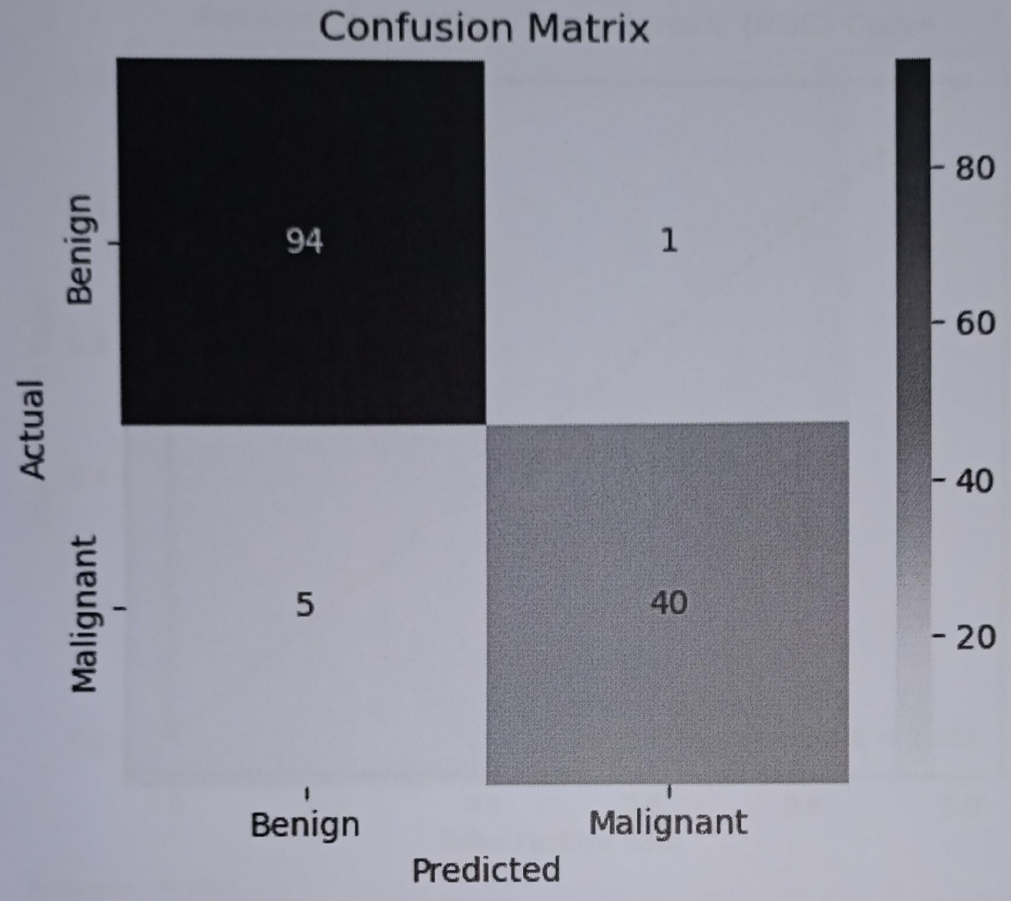
DATE 18/3/25 | PAGE NO. 21 | EXPT. NO. 5

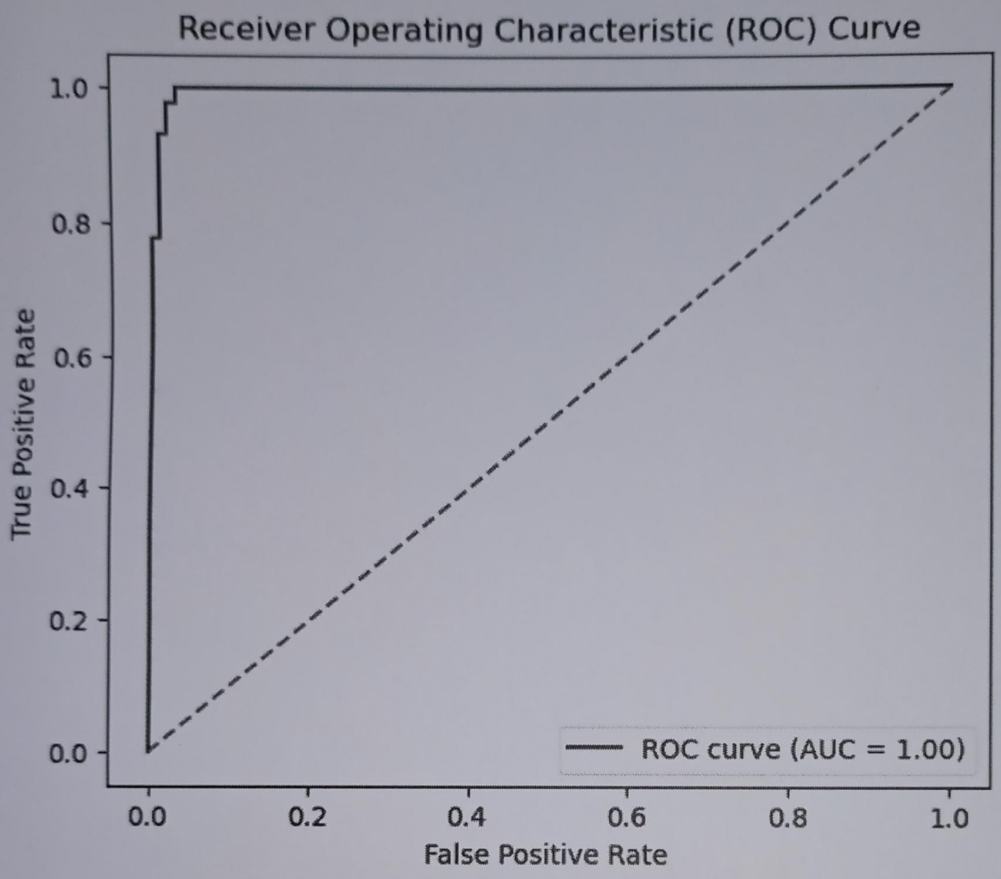
```
y_prob = model.predict_proba(X_test)[:,1]

#Compute Performance Metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

#Print Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

#Plot Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Benign","Malignant"],
            yticklabels=["Benign","Malignant"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

#Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
```

DATE 18/3/25 | PAGE NO. 22 | EXPT. NO. 5

```
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC
        ={roc_auc:.2f})')
plt.plot([0,1],[0,1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()

# Print Evaluation Scores
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"AUC Score: {roc_auc:.4f}")
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ID           699 non-null    int64
 1   Clump        699 non-null    int64
 2   UnifSize     699 non-null    int64
 3   UnifShape    699 non-null    int64
 4   MargAdh      699 non-null    int64
 5   SingEpiSize  699 non-null    int64
 6   BareNuc      699 non-null    object
 7   BlandChrom   699 non-null    int64
 8   NormNucl     699 non-null    int64
 9   Mit          699 non-null    int64
 10  Class        699 non-null    int64
dtypes: int64(10), object(1)
memory usage: 60.2+ KB
None

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97        95
           1       0.98      0.89      0.93        45

    accuracy                           0.96       140
   macro avg       0.96      0.94      0.95       140
weighted avg       0.96      0.96      0.96       140
```


Confusion Matrix
94
1
5
40
Benign
Malignant
Actual
Benign
Malignant
Predicted
80
60
40
20




```
Accuracy: 0.9571
Precision: 0.9756
Recall: 0.8889
F1 Score: 0.9302
AUC Score: 0.9967
```