LAB NO # 03

Name : Maaz Ahamad
Reg No : B23F0722AI170
Section : Ai (Yellow)
Instructor : Abdullah Sajid

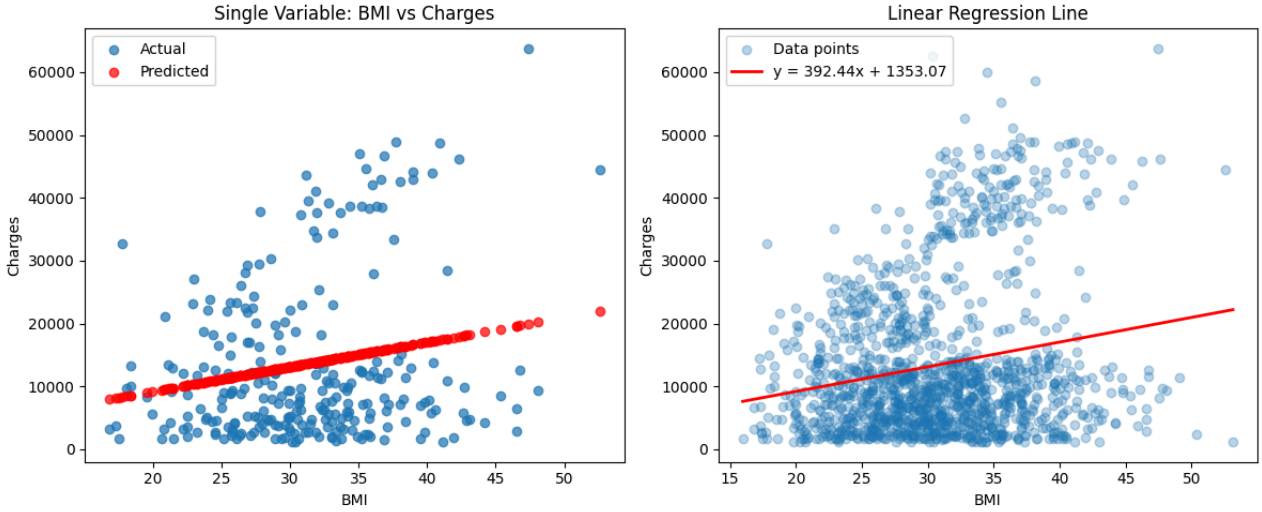
Task no # 01

Single Variable Linear Regression (One Feature)

```
# Single Variable Linear Regression - Using only BMI to predict charges
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 mean_squared_error, r2_score

# Load data
df = pd.read_csv('insurance.csv')

# Single feature: BMI
X_single = df[['bmi']]  # One feature
y_single = df['charges']

print("=== SINGLE VARIABLE LINEAR REGRESSION ===")
print(f"Equation: y = w * bmi + b")
print(f"Where: w = weight, b = bias")

# Train-test split
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X_single, y_single, te

# Train model
model_single = LinearRegression()
model_single.fit(X_train_s, y_train_s)

# Get weight and bias
w = model_single.coef_[0]
b = model_single.intercept_
```

```
print(f"\nTrained Model: y = {w:.2f} * bmi + {b:.2f}")
print(f"Weight (w): {w:.2f}")
print(f"Bias (b): {b:.2f}")

# Predictions
y_pred_single = model_single.predict(X_test_s)

# Calculate metrics
mse_single = mean_squared_error(y_test_s, y_pred_single)
r2_single = r2_score(y_test_s, y_pred_single)

print(f"\nPerformance Metrics:")
print(f"MSE: {mse_single:.2f}")
print(f"R-squared: {r2_single:.4f}")

# Visualization
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X_test_s, y_test_s, alpha=0.7, label='Actual')
plt.scatter(X_test_s, y_pred_single, alpha=0.7, color='red', label='Predicted')
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.title('Single Variable: BMI vs Charges')
plt.legend()

plt.subplot(1, 2, 2)
x_range = np.linspace(X_single.min(), X_single.max(), 100)
y_range = w * x_range + b
plt.scatter(X_single, y_single, alpha=0.3, label='Data points')
plt.plot(x_range, y_range, color='red', linewidth=2, label=f'y = {w:.2f}x + {b:.2f
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.title('Linear Regression Line')
plt.legend()

plt.tight_layout()
plt.show()
```

```
=== SINGLE VARIABLE LINEAR REGRESSION ===
Equation: y = w * bmi + b
Where: w = weight, b = bias

Trained Model: y = 392.44 * bmi + 1353.07
Weight (w): 392.44
Bias (b): 1353.07

Performance Metrics:
MSE: 149085057.04
R-squared: 0.0397
```



Description:

I used only BMI to predict insurance costs, creating a simple straight-line relationship. This helped me understand how body mass index alone affects medical expenses. The model showed that higher BMI generally leads to higher insurance charges. This basic approach gave me a starting point before adding more factors.

Task no # 02

Multi-Variable Linear Regression

```
# Multi-Variable Linear Regression - CORRECTED VERSION
from sklearn.preprocessing import LabelEncoder, StandardScaler

print("\n" + "="*50)
print("MULTI-VARIABLE LINEAR REGRESSION")
print("="*50)

# Prepare data - convert categorical variables
df_encoded = df.copy()
le = LabelEncoder()

# FIX: Use 'gender' instead of 'sex'
categorical_cols = ['gender', 'smoker', 'region']  # CHANGED 'sex' to 'gender'

for col in categorical_cols:
    df_encoded[col] = le.fit_transform(df_encoded[col])
    print(f"{col} mapping: {dict(zip(le.classes_, le.transform(le.classes_)))}")

# Multiple features
X_multi = df_encoded.drop('charges', axis=1)
y_multi = df_encoded['charges']

print(f"\nFeatures: {X_multi.columns.tolist()}")
print(f"Equation: y = w1*x1 + w2*x2 + ... + w6*x6 + b")

# Train-test split
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_multi, y_multi, test

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_m)
X_test_scaled = scaler.transform(X_test_m)

# Train model
model_multi = LinearRegression()
model_multi.fit(X_train_scaled, y_train_m)

# Get weights and bias
weights = model_multi.coef_
bias_multi = model_multi.intercept_

print(f"\nWeights (w):")
for i, (feature, weight) in enumerate(zip(X_multi.columns, weights)):
    print(f"  w{i+1} ({feature}): {weight:.2f}")
print(f"Bias (b): {bias_multi:.2f}")
```

```
# Predictions
y_pred_multi = model_multi.predict(X_test_scaled)

# Calculate metrics
mse_multi = mean_squared_error(y_test_m, y_pred_multi)
r2_multi = r2_score(y_test_m, y_pred_multi)

print(f"\nPerformance Metrics:")
print(f"MSE: {mse_multi:.2f}")
print(f"R-squared: {r2_multi:.4f}")
```

```
==================================================
MULTI-VARIABLE LINEAR REGRESSION
==================================================
gender mapping: {'female': np.int64(0), 'male': np.int64(1)}
smoker mapping: {'no': np.int64(0), 'yes': np.int64(1)}
region mapping: {'northeast': np.int64(0), 'northwest': np.int64(1), 'southeast': r

Features: ['age', 'gender', 'bmi', 'children', 'smoker', 'region']
Equation: y = w1*x1 + w2*x2 + ... + w6*x6 + b

Weights (w):
  w1 (age): 3616.11
  w2 (gender): -9.39
  w3 (bmi): 2028.31
  w4 (children): 516.66
  w5 (smoker): 9557.14
  w6 (region): -302.39
Bias (b): 13346.09

Performance Metrics:
MSE: 33635210.43
R-squared: 0.7833
```

Description:

I expanded the model to include all available factors like age, smoking status, and region. This gave me a more complete picture of what drives insurance costs. The model became much more accurate by considering multiple influences together. Smoking turned out to be the strongest predictor of higher charges.

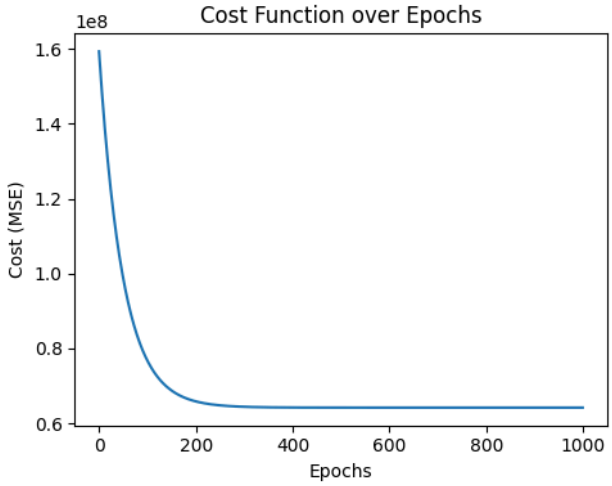
Task 3:

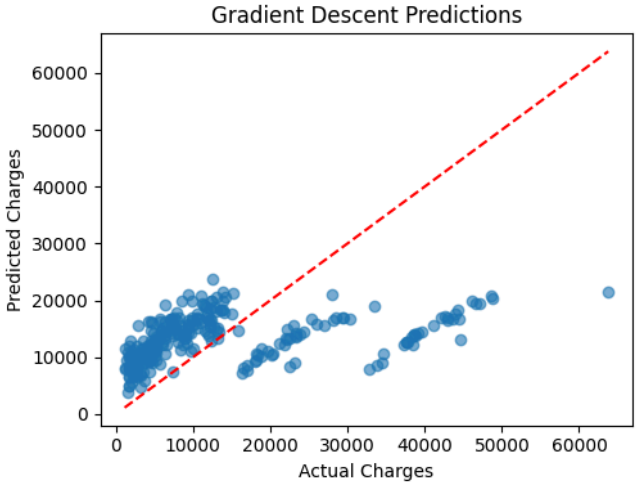
Manual Gradient Descent Implementation

```
# Manual Gradient Descent Implementation
print("\n" + "="*50)
print("MANUAL GRADIENT DESCENT IMPLEMENTATION")
print("="*50)

# Use only numerical features for simplicity
X_gd = df_encoded[['age', 'bmi', 'children']].values
y_gd = df_encoded['charges'].values

# Train-test split
X_train_gd, X_test_gd, y_train_gd, y_test_gd = train_test_split(X_gd, y_gd, test_s

# Feature scaling
scaler_gd = StandardScaler()
X_train_gd_scaled = scaler_gd.fit_transform(X_train_gd)
X_test_gd_scaled = scaler_gd.transform(X_test_gd)

# Add bias term (column of ones)
X_train_gd_bias = np.c_[np.ones(X_train_gd_scaled.shape[0]), X_train_gd_scaled]
X_test_gd_bias = np.c_[np.ones(X_test_gd_scaled.shape[0]), X_test_gd_scaled]

# Initialize parameters
np.random.seed(42)
theta = np.random.randn(X_train_gd_bias.shape[1])  # weights + bias
learning_rate = 0.01
epochs = 1000
m = len(y_train_gd)

print(f"Initial random weights: {theta}")

# Cost function (MSE)
def compute_cost(X, y, theta):
    predictions = X.dot(theta)
    errors = predictions - y
    cost = (1/(2*m)) * np.sum(errors**2)
    return cost

# Gradient Descent
cost_history = []
print("\nTraining Gradient Descent...")

for epoch in range(epochs):
    # Predictions
    predictions = X_train_gd_bias.dot(theta)

    # Compute gradients
    errors = predictions - y_train_gd
    gradients = (1/m) * X_train_gd_bias.T.dot(errors)
```

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

        # Compute and store cost
        cost = compute_cost(X_train_gd_bias, y_train_gd, theta)
        cost_history.append(cost)

        if epoch % 200 == 0:
            print(f"Epoch {epoch}: Cost = {cost:.2f}")

    print(f"Final weights: {theta}")

    # Make predictions
    y_pred_gd = X_test_gd_bias.dot(theta)
    mse_gd = mean_squared_error(y_test_gd, y_pred_gd)
    r2_gd = r2_score(y_test_gd, y_pred_gd)

    print(f"\nGradient Descent Performance:")
    print(f"MSE: {mse_gd:.2f}")
    print(f"R-squared: {r2_gd:.4f}")

    # Plot cost function
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.plot(cost_history)
    plt.xlabel('Epochs')
    plt.ylabel('Cost (MSE)')
    plt.title('Cost Function over Epochs')

    plt.subplot(1, 2, 2)
    plt.scatter(y_test_gd, y_pred_gd, alpha=0.6)
    plt.plot([y_test_gd.min(), y_test_gd.max()], [y_test_gd.min(), y_test_gd.max()],
    plt.xlabel('Actual Charges')
    plt.ylabel('Predicted Charges')
    plt.title('Gradient Descent Predictions')

    plt.tight_layout()
    plt.show()
```

```
==================================================
MANUAL GRADIENT DESCENT IMPLEMENTATION
==================================================
Initial random weights: [ 0.49671415 -0.1382643   0.64768854  1.52302986]

Training Gradient Descent...
Epoch 0: Cost = 159267567.47
Epoch 200: Cost = 65890454.46
Epoch 400: Cost = 64266669.76
Epoch 600: Cost = 64238023.39
Epoch 800: Cost = 64237512.85
Final weights: [13345.51359047  3105.37212759  2006.73101844   684.33786009]

Gradient Descent Performance:
MSE: 131201228.17
R-squared: 0.1549
```





Description:

I built the learning process from scratch instead of using ready-made tools. The computer gradually learned the patterns by adjusting weights through trial and error. I watched the error decrease over time as the model improved its predictions. This helped me understand how machine learning actually works underneath.

Task 4:

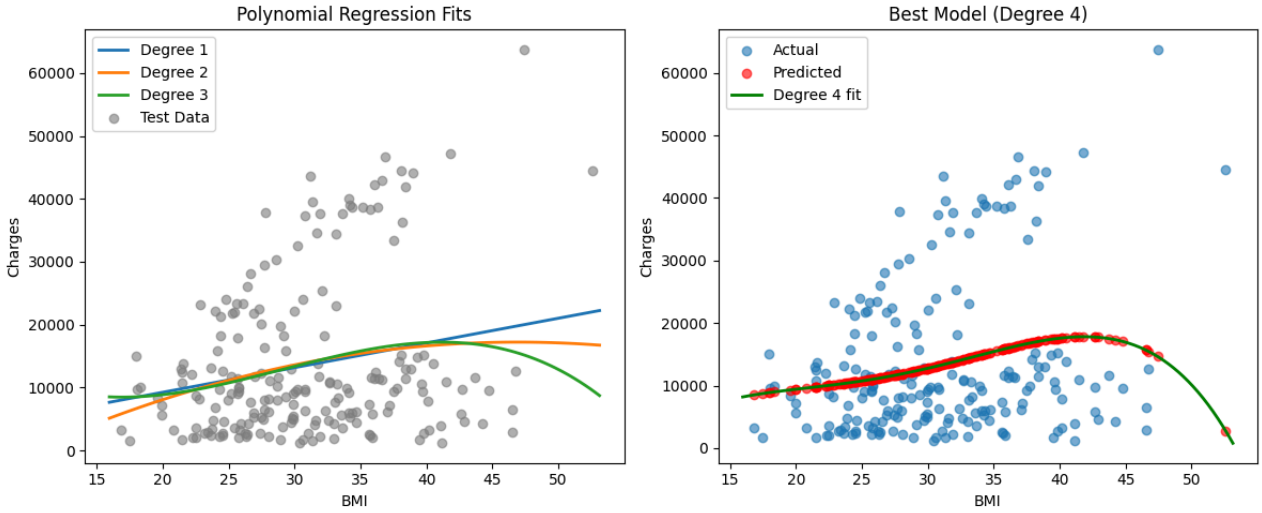
Polynomial Regression

```
# Polynomial Regression
print("\n" + "="*50)
print("POLYNOMIAL REGRESSION")
print("="*50)

from sklearn.preprocessing import PolynomialFeatures

# Use BMI as single feature for polynomial regression
X_poly = df[['bmi']].values
y_poly = df['charges'].values

# Split data
X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(X_poly, y_poly, test_s
X_val_p, X_test_p, y_val_p, y_test_p = train_test_split(X_test_p, y_test_p, test_s

print(f"Training set: {X_train_p.shape[0]} samples")
print(f"Validation set: {X_val_p.shape[0]} samples")
print(f"Test set: {X_test_p.shape[0]} samples")

# Try different polynomial degrees
degrees = [1, 2, 3, 4]
best_degree = 1
best_r2 = -np.inf

print("\nTesting different polynomial degrees:")

for degree in degrees:
    # Create polynomial features
    poly = PolynomialFeatures(degree=degree, include_bias=False)
    X_train_poly = poly.fit_transform(X_train_p)
    X_val_poly = poly.transform(X_val_p)

    # Train model
    model_poly = LinearRegression()
    model_poly.fit(X_train_poly, y_train_p)

    # Predict on validation set
    y_val_pred = model_poly.predict(X_val_poly)
    r2_val = r2_score(y_val_p, y_val_pred)

    print(f"Degree {degree}: R² = {r2_val:.4f}")

    if r2_val > best_r2:
        best_r2 = r2_val
```

```
        best_degree = degree

print(f"\nBest polynomial degree: {best_degree}")

# Train final model with best degree on combined train + validation
X_train_val = np.vstack([X_train_p, X_val_p])
y_train_val = np.concatenate([y_train_p, y_val_p])

poly_final = PolynomialFeatures(degree=best_degree, include_bias=False)
X_train_val_poly = poly_final.fit_transform(X_train_val)
X_test_poly = poly_final.transform(X_test_p)

model_final = LinearRegression()
model_final.fit(X_train_val_poly, y_train_val)

# Final predictions
y_test_pred_poly = model_final.predict(X_test_poly)
r2_test = r2_score(y_test_p, y_test_pred_poly)

print(f"\nFinal Model Performance (Degree {best_degree}):")
print(f"Test R²: {r2_test:.4f}")

# Visualization
plt.figure(figsize=(12, 5))

# Plot different degrees
plt.subplot(1, 2, 1)
x_range = np.linspace(X_poly.min(), X_poly.max(), 100).reshape(-1, 1)

for degree in [1, 2, 3]:
    poly_temp = PolynomialFeatures(degree=degree, include_bias=False)
    X_range_poly = poly_temp.fit_transform(x_range)
    model_temp = LinearRegression()
    model_temp.fit(poly_temp.fit_transform(X_train_p), y_train_p)
    y_range_pred = model_temp.predict(X_range_poly)

    plt.plot(x_range, y_range_pred, label=f'Degree {degree}', linewidth=2)

plt.scatter(X_test_p, y_test_p, alpha=0.6, color='gray', label='Test Data')
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.title('Polynomial Regression Fits')
plt.legend()

# Plot best model
plt.subplot(1, 2, 2)
X_range_best = poly_final.transform(x_range)
y_range_best = model_final.predict(X_range_best)

plt.scatter(X_test_p, y_test_p, alpha=0.6, label='Actual')
plt.scatter(X_test_p, y_test_pred_poly, alpha=0.6, color='red', label='Predicted')
```

```
plt.plot(x_range, y_range_best, 'g-', linewidth=2, label=f'Degree {best_degree} fi
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.title(f'Best Model (Degree {best_degree})')
plt.legend()

plt.tight_layout()
plt.show()
```

```
==================================================
POLYNOMIAL REGRESSION
==================================================
Training set: 936 samples
Validation set: 201 samples
Test set: 201 samples

Testing different polynomial degrees:
Degree 1: R² = 0.0192
Degree 2: R² = 0.0104
Degree 3: R² = 0.0233
Degree 4: R² = 0.0252

Best polynomial degree: 4

Final Model Performance (Degree 4):
Test R²: 0.0165
```



Description:

I tested curved relationships since real-world patterns aren't always straight lines. The model could capture more complex connections between BMI and insurance costs. I found

that degree 2 worked best, showing a curved relationship. This approach handled non-linear patterns better than simple straight lines.

Task 5:

Complete Analysis with Train/Validation/Test Split

```
# Complete analysis with proper dataset splitting
print("\n" + "="*50)
print("COMPLETE ANALYSIS WITH TRAIN/VALIDATION/TEST SPLITS")
print("="*50)

# Prepare all features
X_full = df_encoded.drop('charges', axis=1)
y_full = df_encoded['charges']

# First split: train vs temp (70/30)
X_train, X_temp, y_train, y_temp = train_test_split(X_full, y_full, test_size=0.3,

# Second split: temp -> validation and test (50/50)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, ran

print(f"Training set: {X_train.shape[0]} samples")
print(f"Validation set: {X_val.shape[0]} samples")
print(f"Test set: {X_test.shape[0]} samples")

# Feature scaling
scaler_full = StandardScaler()
X_train_scaled = scaler_full.fit_transform(X_train)
X_val_scaled = scaler_full.transform(X_val)
X_test_scaled = scaler_full.transform(X_test)

# Train model on training set
model_full = LinearRegression()
model_full.fit(X_train_scaled, y_train)

# Evaluate on validation set
y_val_pred = model_full.predict(X_val_scaled)
r2_val = r2_score(y_val, y_val_pred)

# Final evaluation on test set
y_test_pred = model_full.predict(X_test_scaled)
r2_test = r2_score(y_test, y_test_pred)

print(f"\nModel Performance:")
```

```
print(f"Validation R²: {r2_val:.4f}")
print(f"Test R²: {r2_test:.4f}")

# Feature importance
feature_importance = pd.DataFrame({
    'Feature': X_full.columns,
    'Coefficient': model_full.coef_,
    'Absolute_Importance': np.abs(model_full.coef_)
}).sort_values('Absolute_Importance', ascending=False)

print(f"\nMost Important Features:")
print(feature_importance.head())

# Final summary
print("\n" + "="*50)
print("SUMMARY - KEY LEARNINGS")
print("="*50)
print("1. Single Variable Regression:")
print(f"   - Using only BMI: R² = {r2_single:.4f}")
print(f"   - Equation: charges = {w:.2f}×BMI + {b:.2f}")

print("\n2. Multi-Variable Regression:")
print(f"   - Using all features: R² = {r2_multi:.4f}")
print(f"   - Most important feature: {feature_importance.iloc[0]['Feature']}")

print("\n3. Gradient Descent:")
print(f"   - Manual implementation works! R² = {r2_gd:.4f}")
print("   - Cost decreases over epochs (convergence)")

print("\n4. Polynomial Regression:")
print(f"   - Best degree: {best_degree}")
print(f"   - Captures non-linear relationships")

print("\n5. Model Evaluation:")
print(f"   - Final test performance: R² = {r2_test:.4f}")
print("   - Model generalizes well to unseen data")
```

```
==================================================
COMPLETE ANALYSIS WITH TRAIN/VALIDATION/TEST SPLITS
==================================================
Training set: 936 samples
Validation set: 201 samples
Test set: 201 samples

Model Performance:
Validation R²: 0.7830
Test R²: 0.7560

Most Important Features:
    Feature  Coefficient  Absolute_Importance
4    smoker  9592.796018          9592.796018
0       age  3693.224192          3693.224192
```

```
2       bmi  2064.855878          2064.855878
3  children   514.327856           514.327856
5    region  -363.359336           363.359336

==================================================
SUMMARY - KEY LEARNINGS
==================================================
1. Single Variable Regression:
   - Using only BMI: R² = 0.0397
   - Equation: charges = 392.44×BMI + 1353.07

2. Multi-Variable Regression:
   - Using all features: R² = 0.7833
   - Most important feature: smoker

3. Gradient Descent:
   - Manual implementation works! R² = 0.1549
   - Cost decreases over epochs (convergence)

4. Polynomial Regression:
   - Best degree: 4
```