Part 1: Regression Task (California Housing)

Task 1: Load and Split Dataset

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Setting the columns
cols = ["longitude", "latitude", "housingMedianAge", "totalRooms", "totalBedrooms", "population", "households", "medianIncome", "medianHouseValue"]

# Loading the dataset by downloading from "https://s3-eu-west-1.amazonaws.com/pfigshare-u-files/5976036/cal_housing.tgz"
df = pd.read_csv("cal_housing.data", header=None, names=cols)

# Spliting the data into features and label
X = df.drop("medianHouseValue", axis = 1)
y = df["medianHouseValue"]

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

print(f"\nTraining set: {X_train.shape}")
print(f"Test set: {X_test.shape}")
print(f"Features: {X_train.shape[1]}")
```

```
Training set: (16512, 8)
Test set: (4128, 8)
Features: 8
```

Task 2, Step 1: Baseline Model (No Regularization)

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

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

# Observe coefficients
print("Coefficients:")
for i in range(min(5, len(model.coef_))):  # Show first 5
    print(f"  Feature {i}: {model.coef_[i]:.6f}")
print(f"  ... and {len(model.coef_) - 5} more")
print(f"Intercept: {model.intercept_:.6f}")

# Compute MSE
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print(f"\nTraining MSE: {mse_train:.6f}")
print(f"Test MSE: {mse_test:.6f}")
```

```
Coefficients:
  Feature 0: -42632.391717
  Feature 1: -42450.071864
  Feature 2: 1182.809649
  Feature 3: -8.187977
  Feature 4: 116.260128
  ... and 3 more
Intercept: -3578224.234818

Training MSE: 4811134397.884194
Test MSE: 4918556441.477760
```

Task 2, Step 2: Hyperparameter Tuning

```python
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import GridSearchCV

# Define alphas
alphas = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

# Ridge Regression tuning
ridge = Ridge()
ridge_grid = GridSearchCV(ridge, {'alpha': alphas}, cv=5, scoring='neg_mean_squared_error')
ridge_grid.fit(X_train, y_train)

print(f"Ridge - Best alpha: {ridge_grid.best_params_['alpha']}")

# Lasso Regression tuning
lasso = Lasso(max_iter=10000)
lasso_grid = GridSearchCV(lasso, {'alpha': alphas}, cv=5, scoring='neg_mean_squared_error')
lasso_grid.fit(X_train, y_train)

print(f"Lasso - Best alpha: {lasso_grid.best_params_['alpha']}")

# Test set evaluation
ridge_pred = ridge_grid.predict(X_test)
lasso_pred = lasso_grid.predict(X_test)

ridge_mse = mean_squared_error(y_test, ridge_pred)
lasso_mse = mean_squared_error(y_test, lasso_pred)

print(f"\nTest MSE - Ridge: {ridge_mse:.6f}")
print(f"Test MSE - Lasso: {lasso_mse:.6f}")
```

```
Ridge - Best alpha: 10
Lasso - Best alpha: 10

Test MSE - Ridge: 4918567284.465969
Test MSE - Lasso: 4918555581.562370
```

Task 2, Step 3: Regularization Experiments (L1 vs L2)

```python
# Train models with best parameters
ridge_best = Ridge(alpha=ridge_grid.best_params_['alpha'])
lasso_best = Lasso(alpha=lasso_grid.best_params_['alpha'], max_iter=10000)

ridge_best.fit(X_train, y_train)
lasso_best.fit(X_train, y_train)

# Compare coefficients
print("\nCoefficient Comparison (first 8 features):")
print("Feature\tBaseline\t\tRidge\t\t\tLasso")
for i in range(8):
    print(f"{i}\t{model.coef_[i]:.6f}\t{ridge_best.coef_[i]:.6f}\t{lasso_best.coef_[i]:.6f}")

# Count zero coefficients
zero_lasso = sum(lasso_best.coef_ == 0)
print(f"\nZero coefficients in Lasso: {zero_lasso}/{len(lasso_best.coef_)}")

# Performance comparison
ridge_train_mse = mean_squared_error(y_train, ridge_best.predict(X_train))
ridge_test_mse = mean_squared_error(y_test, ridge_best.predict(X_test))

lasso_train_mse = mean_squared_error(y_train, lasso_best.predict(X_train))
lasso_test_mse = mean_squared_error(y_test, lasso_best.predict(X_test))

print(f"\nPerformance Comparison:")
print(f"{'Model':<10} {'Train MSE':<15} {'Test MSE':<15}")
print("-" * 40)
print(f"{'Baseline':<10} {mse_train:<15.6f} {mse_test:<15.6f}")
print(f"{'Ridge':<10} {ridge_train_mse:<15.6f} {ridge_test_mse:<15.6f}")
print(f"{'Lasso':<10} {lasso_train_mse:<15.6f} {lasso_test_mse:<15.6f}")
```

```python
print("\nDiscussion:")
print("1. L1 produces sparse coefficients (feature selection)")
print("2. L2 shrinks coefficients without zeroing them")
print("3. Regularization reduces variance, prevents overfitting")
print("4. Excessive regularization increases bias")
```

```
Coefficient Comparison (first 8 features):
Feature Baseline                Ridge                   Lasso
0       -42632.391717   -42535.627082   -42595.282794
1       -42450.071864   -42359.666504   -42415.402048
2       1182.809649     1184.351988     1183.328980
3       -8.187977       -8.196936       -8.191118
4       116.260128      116.124492      116.204998
5       -38.492213      -38.496151      -38.493803
6       46.342572       46.569026       46.430671
7       40538.404387    40543.565513    40540.088671

Zero coefficients in Lasso: 0/8

Performance Comparison:
Model      Train MSE       Test MSE
----------------------------------------
Baseline   4811134397.884194 4918556441.477760
Ridge      4811139082.000711 4918567284.465969
Lasso      4811135093.259238 4918555581.562370

Discussion:
1. L1 produces sparse coefficients (feature selection)
2. L2 shrinks coefficients without zeroing them
3. Regularization reduces variance, prevents overfitting
4. Excessive regularization increases bias
```

PART 2: CLASSIFICATION TASK (Breast Cancer)

Task 1: Load and Split Dataset¶

```
from sklearn.datasets import load_breast_cancer

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Training set size: {X_train.shape}")
print(f"Test set size: {X_test.shape}")
```

```
Training set size: (455, 30)
Test set size: (114, 30)
```

Task 2, Step 1: Baseline Model (No Regularization)

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

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

# Observe coefficients
print("Coefficients:")
for i in range(min(5, len(model.coef_))):  # Show first 5
    print(f"  Feature {i}: {model.coef_[i]:.6f}")
print(f"  ... and {len(model.coef_) - 5} more")
print(f"Intercept: {model.intercept_:.6f}")

# Compute MSE
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print(f"\nTraining MSE: {mse_train:.6f}")
print(f"Test MSE: {mse_test:.6f}")
```

```
Baseline Logistic Regression
Train Accuracy: 0.9626373626373627
Test Accuracy: 0.956140350877193
Coefficients: [[ 0.98208299  0.22519686 -0.36688444  0.0262268  -0.15507824 -0.22867976
  -0.52338614 -0.2793554  -0.22391176 -0.03605388 -0.09476544  1.39135347
  -0.16429246 -0.08903006 -0.02250974  0.04944847 -0.04186075 -0.03193634
  -0.03298528  0.01189208  0.10400464 -0.51389384 -0.01711567 -0.01662253
  -0.30695364 -0.75341491 -1.41533107 -0.50382259 -0.73542849 -0.09913574]]
```

Task 2, Step 2: Hyperparameter Tuning

```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'solver': ['liblinear']}
log_cv = GridSearchCV(log_reg, param_grid, scoring = 'accuracy', cv = 5)
log_cv.fit(X_train, y_train)

print("Best Parameters:", log_cv.best_params_)

best_log = log_cv.best_estimator_

y_train_pred = best_log.predict(X_train)
y_test_pred = best_log.predict(X_test)

print("Tuned Logistic Regression")
print("Train Accuracy:", accuracy_score(y_train, y_train_pred))
print("Test Accuracy:", accuracy_score(y_test, y_test_pred))

# L1 Logistic Regression
log_l1 = LogisticRegression(C=log_cv.best_params_['C'], l1_ratio = 1.0, solver='liblinear', max_iter=10000)
log_l1.fit(X_train, y_train)

# L2 Logistic Regression
log_l2 = LogisticRegression(C=log_cv.best_params_['C'], l1_ratio = 0.0, solver='liblinear', max_iter=10000)
log_l2.fit(X_train, y_train)

print("L1 Train Accuracy:", accuracy_score(y_train, log_l1.predict(X_train)))
print("L1 Test Accuracy:", accuracy_score(y_test, log_l1.predict(X_test)))

print("L2 Train Accuracy:", accuracy_score(y_train, log_l2.predict(X_train)))
print("L2 Test Accuracy:", accuracy_score(y_test, log_l2.predict(X_test)))
```

```
Best Parameters: {'C': 10, 'solver': 'liblinear'}
Tuned Logistic Regression
Train Accuracy: 0.9692307692307692
Test Accuracy: 0.956140350877193
L1 Train Accuracy: 0.9824175824175824
L1 Test Accuracy: 0.9736842105263158
L2 Train Accuracy: 0.9692307692307692
L2 Test Accuracy: 0.956140350877193
```

Task 2, Step 3: Regularization Experiments (L1 vs L2)

```
import matplotlib.pyplot as plt

# Coefficients
coeff_l1 = pd.Series(log_l1.coef_[0], index=load_breast_cancer().feature_names)
coeff_l2 = pd.Series(log_l2.coef_[0], index=load_breast_cancer().feature_names)

print("L1 coefficients:")
print(coeff_l1)

print("\nL2 coefficients:")
print(coeff_l2)

# Count number of non-zero coefficients
print("\nNon-zero L1 coefficients:", np.sum(coeff_l1 != 0))
print("Non-zero L2 coefficients:", np.sum(coeff_l2 != 0))

print("Accuracy Comparison: ")
results = pd.DataFrame({
    'Model': ['L1', 'L2'],
    'Train Accuracy': [
        accuracy_score(y_train, log_l1.predict(X_train)),
        accuracy_score(y_train, log_l2.predict(X_train))
    ],
    'Test Accuracy': [
        accuracy_score(y_test, log_l1.predict(X_test)),
        accuracy_score(y_test, log_l2.predict(X_test))
    ]
})

print(results)
```

```
plt.figure(figsize=(12,5))

plt.plot(log_l1.coef_[0], label="L1")
plt.plot(log_l2.coef_[0], label="L2")

plt.legend()
plt.title("Logistic Regression Coefficients (L1 vs L2)")
plt.show()
```

```
L1 coefficients:
mean radius                1.520500
mean texture               0.179126
mean perimeter            -0.035877
mean area                 -0.011367
mean smoothness            0.000000
mean compactness           0.000000
mean concavity             0.000000
mean concave points      -14.958532
mean symmetry              0.000000
mean fractal dimension     0.000000
radius error               0.000000
texture error              3.210841
perimeter error           -0.867325
area error                -0.091785
smoothness error           0.000000
compactness error          0.000000
concavity error            2.399447
concave points error       0.000000
symmetry error             0.000000
fractal dimension error    0.000000
worst radius               0.864906
worst texture             -0.599582
worst perimeter            0.133197
worst area                -0.026773
worst smoothness           0.000000
worst compactness          0.000000
worst concavity           -2.313812
worst concave points     -30.989909
worst symmetry            -6.643579
worst fractal dimension    0.000000
dtype: float64
```

```
L2 coefficients:
mean radius                4.398356
mean texture               0.296195
mean perimeter            -0.498847
mean area                 -0.008948
mean smoothness           -0.546015
mean compactness          -0.819821
mean concavity            -1.604845
mean concave points       -1.233964
mean symmetry             -0.739526
mean fractal dimension    -0.022000
radius error              -0.321500
texture error              3.604604
perimeter error           -0.969272
area error                -0.071975
smoothness error          -0.074886
compactness error          0.424390
concavity error            0.441039
concave points error      -0.100238
symmetry error            -0.080574
fractal dimension error    0.084301
worst radius               0.398100
worst texture             -0.672205
worst perimeter            0.188310
worst area                -0.027294
worst smoothness          -1.040566
worst compactness         -2.185084
worst concavity           -3.241211
worst concave points      -2.042643
worst symmetry            -2.605688
worst fractal dimension   -0.153856
dtype: float64
```

```
Non-zero L1 coefficients: 16
Non-zero L2 coefficients: 30
Accuracy Comparison:
  Model  Train Accuracy  Test Accuracy
0    L1        0.982418       0.973684
1    L2        0.969231       0.956140
```