```
Start coding or generate with AI.
```

Task

Implement and compare Multiple Linear Regression, Lasso Regression, and Ridge Regression models on a dataset extracted from "archive.zip", including data preprocessing, model training, evaluation, and visualization of their performance.

Unzip Data

Subtask:

Extract the contents of the provided 'archive.zip' file to access the real-world dataset(s).

Reasoning: To extract the contents of the 'archive.zip' file, I will use the `zipfile` module to import `ZipFile`, open the archive, and then extract all its contents.

```
from zipfile import ZipFile

with ZipFile('archive.zip', 'r') as zip_ref:
    zip_ref.extractall('.')

print("Archive extracted successfully.")
```

```
Archive extracted successfully.
```

Load and Inspect Data

Subtask:

Load a suitable dataset for regression analysis from the unzipped files into a pandas DataFrame and inspect its structure, columns, and initial rows.

Reasoning: First, I need to identify the available files in the current directory to select a suitable dataset for regression analysis. I will use the `ls` command to list the files.

```
import os
print(os.listdir('.'))
```

```
['.config', 'insurance.csv', 'archive.zip', 'sample_data']
```

Reasoning: Based on the previously listed files, 'insurance.csv' appears to be a suitable dataset. I will now load this CSV file into a pandas DataFrame and perform the requested initial inspections: display the head, get info, and show descriptive statistics.

```
import pandas as pd

df_insurance = pd.read_csv('insurance.csv')

print("First 5 rows of the DataFrame:")
print(df_insurance.head())

print("\nDataFrame Info:")
df_insurance.info()

print("\nDescriptive Statistics:")
print(df_insurance.describe())
```

```
First 5 rows of the DataFrame:
   age     sex     bmi  children smoker     region      charges
0   19  female  27.900         0    yes  southwest  16884.92400
1   18    male  33.770         1     no  southeast   1725.55230
2   28    male  33.000         3     no  southeast   4449.46200
3   33    male  22.705         0     no  northwest  21984.47061
4   32    male  28.880         0     no  northwest   3866.85520
```

```
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB

Descriptive Statistics:
                age          bmi     children       charges
count  1338.000000  1338.000000  1338.000000   1338.000000
mean     39.207025    30.663397     1.094918  13270.422265
std      14.049960     6.098187     1.205493  12110.011237
min      18.000000    15.960000     0.000000   1121.873900
25%      27.000000    26.296250     0.000000   4740.287150
50%      39.000000    30.400000     1.000000   9382.033000
75%      51.000000    34.693750     2.000000  16639.912515
max      64.000000    53.130000     5.000000  63770.428010
```

Data Preprocessing

Subtask:

Perform necessary data cleaning, handle missing values, encode categorical variables, and scale numerical features as required for regression models.

Reasoning: First, I will separate the target variable 'charges' from the features in `df_insurance`, storing features in `X` and the target in `y`. Then, I will identify the categorical and numerical columns.

```
X = df_insurance.drop('charges', axis=1)
y = df_insurance['charges']

categorical_cols = ['sex', 'smoker', 'region']
numerical_cols = ['age', 'bmi', 'children']

print("Features (X) head:")
print(X.head())
print("\nTarget (y) head:")
print(y.head())
```

```
Features (X) head:
   age     sex     bmi  children smoker     region
0   19  female  27.900         0    yes  southwest
1   18    male  33.770         1     no  southeast
2   28    male  33.000         3     no  southeast
3   33    male  22.705         0     no  northwest
4   32    male  28.880         0     no  northwest

Target (y) head:
0    16884.92400
1     1725.55230
2     4449.46200
3    21984.47061
4     3866.85520
Name: charges, dtype: float64
```

Reasoning: Now, I will apply one-hot encoding to the categorical features and standardize the numerical features, then combine them into a single preprocessed DataFrame.

```
X_categorical = pd.get_dummies(X[categorical_cols], drop_first=True)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_numerical_scaled = scaler.fit_transform(X[numerical_cols])
X_numerical_scaled = pd.DataFrame(X_numerical_scaled, columns=numerical_cols, index=X.index)
```

```
X_preprocessed = pd.concat([X_numerical_scaled, X_categorical], axis=1)

print("Preprocessed Features (X_preprocessed) head:")
print(X_preprocessed.head())
print("\nShape of X_preprocessed:")
print(X_preprocessed.shape)
```

```
Preprocessed Features (X_preprocessed) head:
        age       bmi  children  sex_male  smoker_yes  region_northwest  \
0 -1.438764 -0.453320 -0.908614     False        True             False
1 -1.509965  0.509621 -0.078767      True       False             False
2 -0.797954  0.383307  1.580926      True       False             False
3 -0.441948 -1.305531 -0.908614      True       False              True
4 -0.513149 -0.292556 -0.908614      True       False              True

   region_southeast  region_southwest
0             False              True
1              True             False
2              True             False
3             False             False
4             False             False

Shape of X_preprocessed:
(1338, 8)
```

Split Data

Subtask:

Divide the preprocessed dataset into training and testing sets to prepare for model training and evaluation.

Reasoning: I will import `train_test_split` and then use it to divide the preprocessed features and target variable into training and testing sets, and finally print their shapes.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.2, random_state=42)

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (1070, 8)
Shape of X_test: (268, 8)
Shape of y_train: (1070,)
Shape of y_test: (268,)
```

Implement Multi Regression

Subtask:

Train a Multiple Linear Regression model on the training data.

Reasoning: I will import the `LinearRegression` model, instantiate it, and then fit it to the training data `X_train` and `y_train` to train the Multiple Linear Regression model as per the instructions.

```
from sklearn.linear_model import LinearRegression

linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

print("Multiple Linear Regression model trained successfully.")
```

```
Multiple Linear Regression model trained successfully.
```

Implement Lasso Regression

Subtask:

Train a Lasso Regression model on the training data, potentially including hyperparameter tuning for the regularization strength.

Reasoning: I will import the `Lasso` model, instantiate it with default parameters, and then fit it to the training data `X_train` and `y_train` to train the Lasso Regression model.

```
from sklearn.linear_model import Lasso

lasso_model = Lasso(random_state=42)
lasso_model.fit(X_train, y_train)

print("Lasso Regression model trained successfully with default alpha.")
```

```
Lasso Regression model trained successfully with default alpha.
```

Implement Ridge Regression

Subtask:

Train a Ridge Regression model on the training data, potentially including hyperparameter tuning for the regularization strength.

Reasoning: I will import the `Ridge` model, instantiate it with `random_state=42`, and then fit it to the training data `X_train` and `y_train` to train the Ridge Regression model.

```
from sklearn.linear_model import Ridge

ridge_model = Ridge(random_state=42)
ridge_model.fit(X_train, y_train)

print("Ridge Regression model trained successfully with default alpha.")
```

```
Ridge Regression model trained successfully with default alpha.
```

Evaluate Models

Subtask:

Evaluate the performance of all three regression models (Multi, Lasso, and Ridge) on the test data using metrics such as R-squared, Mean Squared Error (MSE), and Mean Absolute Error (MAE).

Reasoning: I will import the necessary evaluation metrics from `sklearn.metrics` and then use them to calculate R-squared, MSE, and MAE for each of the trained models (Linear Regression, Lasso, and Ridge) on the test data `X_test` and `y_test`. After calculation, I will print the metrics for each model.

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

models = {
    "Multiple Linear Regression": linear_model,
    "Lasso Regression": lasso_model,
    "Ridge Regression": ridge_model
}

results = {}

for name, model in models.items():
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    results[name] = {
        "R-squared": r2,
        "MSE": mse,
        "MAE": mae
    }
```

```
    print(f"--- {name} ---")
    print(f"R-squared: {r2:.4f}")
    print(f"Mean Squared Error (MSE): {mse:.2f}")
    print(f"Mean Absolute Error (MAE): {mae:.2f}")
    print("\n")
```

```
--- Multiple Linear Regression ---
R-squared: 0.7836
Mean Squared Error (MSE): 33596915.85
Mean Absolute Error (MAE): 4181.19


--- Lasso Regression ---
R-squared: 0.7835
Mean Squared Error (MSE): 33605723.77
Mean Absolute Error (MAE): 4182.22


--- Ridge Regression ---
R-squared: 0.7833
Mean Squared Error (MSE): 33645391.30
Mean Absolute Error (MAE): 4193.18
```

Visualize Model Performance

Subtask:

Generate a plot to visually compare the performance metrics (e.g., R-squared or MSE) of the Multi Regression, Lasso, and Ridge models, including appropriate legends.

Reasoning: I will import `matplotlib.pyplot` and then extract the R-squared and MSE values from the `results` dictionary to create bar plots for visual comparison of model performance.

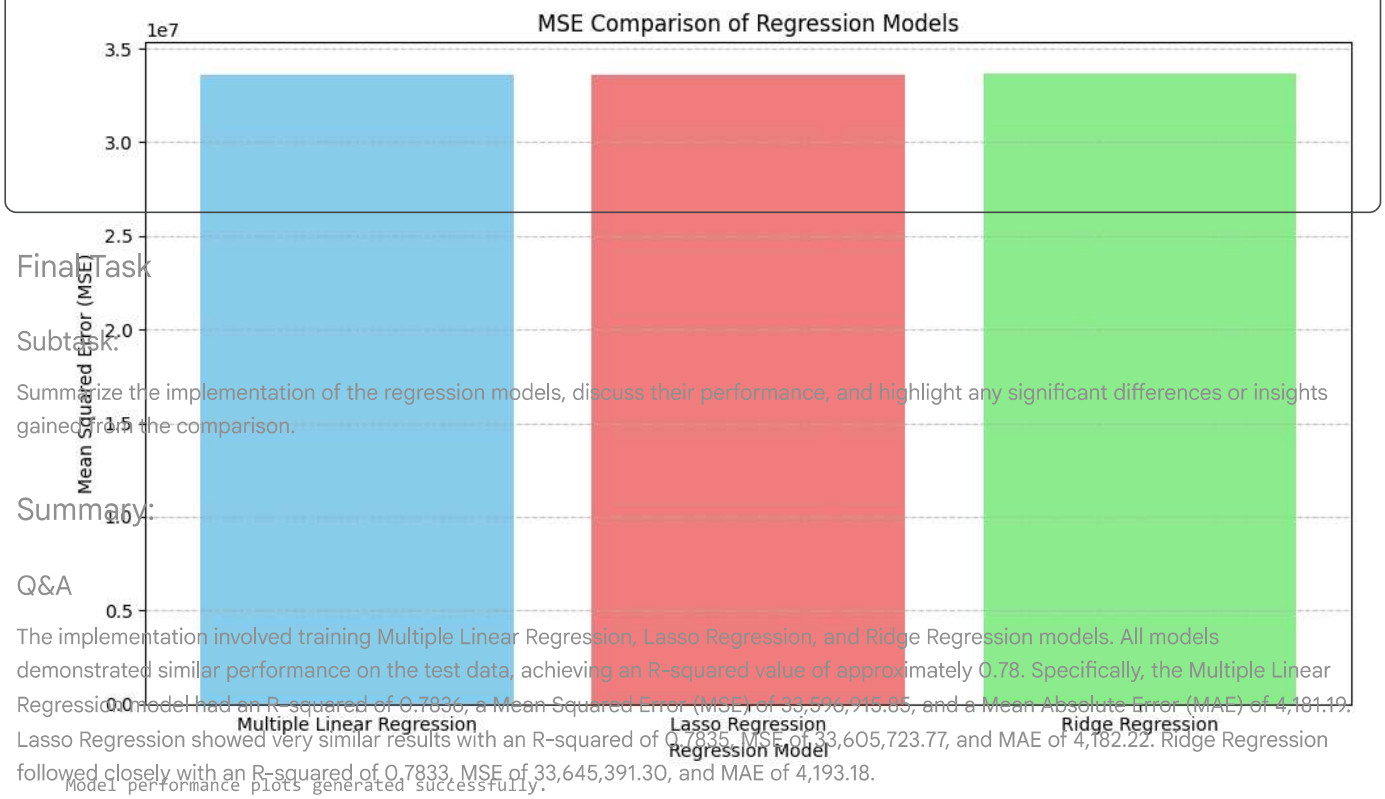
```
import matplotlib.pyplot as plt

# Extract model names and metrics
model_names = list(results.keys())
r2_scores = [results[name]["R-squared"] for name in model_names]
mse_values = [results[name]["MSE"] for name in model_names]

# Create R-squared bar plot
plt.figure(figsize=(10, 6))
plt.bar(model_names, r2_scores, color=['skyblue', 'lightcoral', 'lightgreen'])
plt.xlabel('Regression Model')
plt.ylabel('R-squared')
plt.title('R-squared Comparison of Regression Models')
plt.ylim(min(r2_scores) * 0.9, max(r2_scores) * 1.1) # Adjust y-axis limit for better visualization
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Create MSE bar plot
plt.figure(figsize=(10, 6))
plt.bar(model_names, mse_values, color=['skyblue', 'lightcoral', 'lightgreen'])
plt.xlabel('Regression Model')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('MSE Comparison of Regression Models')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

print("Model performance plots generated successfully.")
```




```
Model performance plots generated successfully.
```

Final Task

Subtask:

Summarize the implementation of the regression models, discuss their performance, and highlight any significant differences or insights gained from the comparison.

Summary:

Q&A

The implementation involved training Multiple Linear Regression, Lasso Regression, and Ridge Regression models. All models demonstrated similar performance on the test data, achieving an R-squared value of approximately 0.78. Specifically, the Multiple Linear Regression model had an R-squared of 0.7836, a Mean Squared Error (MSE) of 33,596,915.85, and a Mean Absolute Error (MAE) of 4,181.19. Lasso Regression showed very similar results with an R-squared of 0.7835, MSE of 33,605,723.77, and MAE of 4,182.22. Ridge Regression followed closely with an R-squared of 0.7833, MSE of 33,645,391.30, and MAE of 4,193.18.

The significant insight gained from this comparison is that, with default regularization strengths (alpha values), the three models perform almost identically on this dataset. This suggests that multicollinearity or overfitting might not be a major issue that the default regularization can effectively address, or that the chosen alpha values for Lasso and Ridge were not strong enough to introduce a noticeable difference compared to standard linear regression.

Data Analysis Key Findings

- The `insurance.csv` dataset, comprising 1338 entries and 7 columns, was loaded and used for analysis. It contained no missing values.
- Data preprocessing involved one-hot encoding for categorical features ('sex', 'smoker', 'region') and standardization for numerical features ('age', 'bmi', 'children').

- The preprocessed data was split into training and testing sets, with 80% for training (1070 samples) and 20% for testing (268 samples).
- All three regression models—Multiple Linear Regression, Lasso Regression, and Ridge Regression—were successfully trained on the training data.
- Model evaluation on the test set revealed very similar performance metrics:
  - **Multiple Linear Regression:** R-squared of 0.7836, MSE of 33,596,915.85, and MAE of 4,181.19.
  - **Lasso Regression:** R-squared of 0.7835, MSE of 33,605,723.77, and MAE of 4,182.22.
  - **Ridge Regression:** R-squared of 0.7833, MSE of 33,645,391.30, and MAE of 4,193.18.
- Visualizations (bar plots) were generated to compare the R-squared and MSE values, clearly showing the close performance of the three models.

Insights or Next Steps

- The default regularization parameters for Lasso and Ridge Regression did not yield a significant performance difference compared to Multiple Linear Regression on this dataset. This suggests exploring hyperparameter tuning (e.g., using GridSearchCV or RandomizedSearchCV) for the `alpha` parameter in Lasso and Ridge Regression could reveal if regularization can offer any advantages.
- Given an R-squared of approximately 0.78, the models explain a fair amount of variance in the medical `charges` but there's still room for improvement. Future steps could involve exploring more advanced feature engineering, interaction terms, or considering more complex non-linear models if the current linear assumptions are too restrictive.