Short Assignment 1

This is an individual assignment.

Objectives

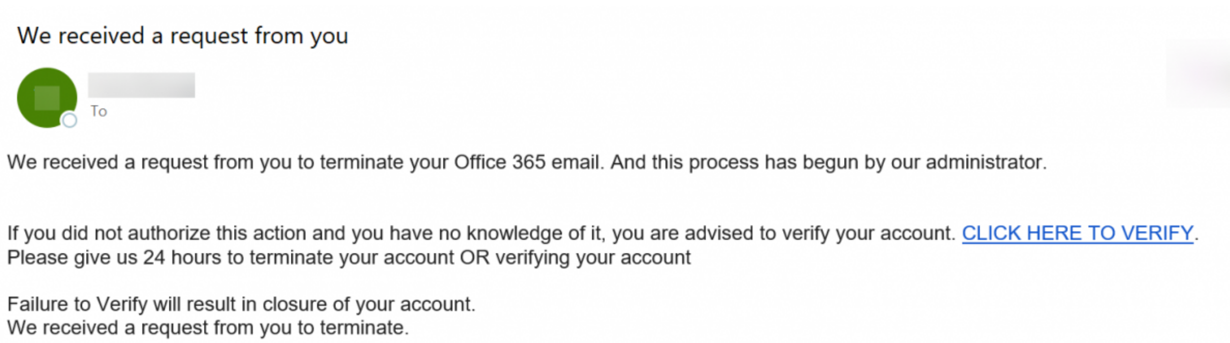
- Define basic terminology in supervised learning algorithms.
- Develop a supervised learning algorithm for regression tasks.
- Implement linear regression model with Python code.
- Utilize trained model for making predictions and measure performance.

Question 1 (1 point)

Assume we are given the task of building a system to distinguish an email as spam or ham (not spam).

In [ ]:
```
from IPython.display import Image
Image('figures/spam_email.png', width=900)
```

Out[ ]:


We received a request from you

To

We received a request from you to terminate your Office 365 email. And this process has begun by our administrator.

If you did not authorize this action and you have no knowledge of it, you are advised to verify your account. CLICK HERE TO VERIFY.
Please give us 24 hours to terminate your account OR verifying your account

Failure to Verify will result in closure of your account.
We received a request from you to terminate.

Use the fundamental components of a supervised learning system to illustrate how you would design a system to solve this task. Explain the role of each component in the system in accomplishing the task of classifying an email as spam or ham.

To design a supervised learning process to classify email as spam or ham (not spam), we can divide the task into its main components and define the role of each component in performing the classification task Basic components with structure in such cases like:

1. **Data Collection and Preprocessing :**

    - Role: Collect a labeled dataset of emails where each email is labeled as either spam or ham. This dataset serves as the training data.

- Explanation: The training data is essential for the supervised learning process. It helps the model learn patterns and features that distinguish between spam and ham emails. Preprocessing steps may include tokenization, lowercasing, and removing stop words.

2. **Feature Extraction:**

   - Role: Convert the email text into numerical features that can be used by the machine learning algorithm.
   - Explanation: Feature extraction involves transforming the text data into a format that can be used for modeling. Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency) vectorization, word embeddings, and feature engineering.

3. **Model Selection:**

   - Role: Choose an appropriate supervised learning algorithm or model.
   - Explanation: Select a machine learning algorithm that is well-suited for text classification tasks like Naive Bayes, Support Vector Machines, or deep learning models like Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs). The choice of model depends on the dataset and task complexity.

4. **Training the Model:**

   - Role: Trains the selected model on the labeled training dataset.
   - Explanation: The model will learns to recognize patterns and associations between features and labels in the training data. During training, the model adjusts its parameters to minimize the clasification error.

5. **Model Evaluation:**

   - Role: Assess the model's performance using evaluation metrics.
   - Explanation: Use metrics like accuracy, precision, recall, and F1-score to evaluate how well the model performs on a separate validation or test dataset. This step helps ensure the model's generalization to unseen data.

6. **Hyperparameter Tuning:**

   - Role: Optimizes the model's hyperparameters.
   - Explanation: Will do experiments with different hyperparameter settings to improve the model's performance. Techniques like cross-validation can be used to find the best hyperparameters.

7. **Deployment:**

   - Role: Deploies the trained model to classify the incoming emails.
   - Explanation: Once the model has been trained and evaluated as a good performance, it will deploy it to classify incoming emails. Emails can be automatically classified as spam or ham based on the model's predictions.

8. **Feedback Loop:**

   - Role: Collects users feedbacks to improve the model.

- Explanation: Will monitor the model's performance in a real-world setting and collects the users feedbacks. This feedbacks can be used to retrain the model periodically, improving its accuracy over time.

9. **Scalability and Maintenance:**

    - Role: Ensures the system scales to handle a large volume of emails and maintain the model's performance.
    - Explanation: As the volume of emails grows, the system should be scalable and capable of handling increased loads. Regular maintenance is required to update the model with new data and retrain it as email patterns change.

By integrating these fundamental components into a supervised learning system, we can effectively classify incoming emails as either spam or ham and protect users from unwanted and potentially harmful messages.

Question 2 (1.5 points)

Consider the training set with N data points $\{x_i\}_{i=1}^N$, where $x_i \in \mathbb{R}$, and its corresponding target labels $\{t_i\}_{i=1}^N$, where $t_i \in \mathbb{R}$. Consider the feature representation

$$\phi(x) = [1, \phi_1(x), \phi_2(x), \ldots, \phi_M(x)]^T$$

where $\phi_i(x)$ is a Gaussian basis function defined as:

$$\phi_i(x) = \exp(-\gamma_i(x-\mu_i)^2), \quad i = 1, \ldots, M$$

with μ_i as the center of component i and γ_i the precision (inverse of variance) of component i.

Answer the following questions:

1. (0.5 points) **Write down the feature matrix $\mathbf{X}$ for $M=2$. Keep your notation neat.**

For $(M=2)$, the feature matrix $(\mathbf{X})$ would be constructed using the Gaussian basis functions as bellow:

$$\mathbf{X} = \begin{bmatrix} 1 & \phi_1(x_1) & \phi_2(x_1) \\ 1 & \phi_1(x_2) & \phi_2(x_2) \\ \vdots & \vdots & \vdots \\ 1 & \phi_1(x_N) & \phi_2(x_N) \end{bmatrix}$$

Where:

$$(\phi_1(x_i) = \exp(-\gamma_1(x_i - \mu_1)^2)) \ for(i = 1, 2, \ldots, N)$$

$$(\phi_2(x_i) = \exp(-\gamma_2(x_i - \mu_2)^2)) \ for(i = 1, 2, \ldots, N)$$

Each row of $(\mathbf{X})$ corresponds to a data point (x_i) and contains the basis functions $(\phi_1(x_i))$ and $(\phi_2(x_i))$.

1. (0.5 points) **What are the hyperparameters in this problem?**

In this problem, the hyperparameters are the parameters that are set before training the model and are not learned from the data. For this problem, the hyperparameters are:

1. M: The number of basis functions. It determines the complexity of the model. Larger values of (M) result in more complex models.
2. μ_i: The centers of the Gaussian basis functions. Each μ_i is a hyperparameter that determines where the basis functions are centered in the feature space.
3. γ_i: The precision (inverse of variance) of the Gaussian basis functions. Each γ_i is a hyperparameter that controls the width of the basis functions.

These hyperparameters are set prior to training the model and play a crucial role in determining the model's capacity to fit the data and its ability to generalize to new data. Proper tuning of these hyperparameters is essential for achieving good model performance. In contrast, the parameter w are learned from the training data during the model training process so it is not a hyperparameter here.

1. (0.5 points) **Suppose you want to minimize the *squared error* with a *Lasso regularizer*. Write down the objective function.**

The objective function for minimizing the squared error with a Lasso (L1) regularizer is commonly known as the Lasso regression objective function. It is a kind of penalty function. It combines the squared error term (which measures the fit to the data) with a regularization term. The objective function for Lasso regression is defined as bellow:

$$\text{Lasso Objective Function} = \sum_{i=1}^{N}(t_i - \mathbf{w}^T\boldsymbol{\phi}(x_i))^2 + \lambda\sum_{j=1}^{M}|w_j|$$

Where:

- (N) is the number of data points.
- (M) is the number of features or basis functions.
- (t_i) is the target value for the (i)-th data point.
- $(\mathbf{w})$ is the vector of model coefficients to be learned.
- $(\boldsymbol{\phi}(x_i))$ is the feature vector for the (i)-th data point, including basis functions.
- (w_j) is the (j)-th coefficient in the vector $(\mathbf{w})$.

- (λ) is the regularization parameter, which controls the strength of the L1 regularization term.

The first term in the objective function represents the squared error or the sum of squared differences between the predicted values $(\mathbf{w}^T \boldsymbol{\phi}(x_i))$ and the actual target values (t_i) for all data points. This term will help the model for fitting for the training data.

The second term represents the L1 regularization term, which is the sum of the absolute values of the coefficients (w_j). This term, effectively help some coefficients to get close to the zero in which will lead to feature selection.

The goal of Lasso regression is to find the values of the coefficients $(\mathbf{w})$ that minimize this objective function, effectively balancing between fitting the data and achieving sparsity in the model. The regularization parameter (λ) controls the trade-off between these two objectives.

Question 3 (1 point)

Suppose that you are working with a two-dimensional features space, where x_1 and x_2 are the two features. Upon receiving a sample $\mathbf{x} = [x_1, x_2]^T$, the goal is to predict a continuous value t. Assume that we have examples of training pairs $\{(x_1, x_2)_i, t_i\}_{i=1}^N$. Suppose we want to make a quadratic mapper function of the form,

$$f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 (x_1)^2 + w_5 (x_2)^2$$

Explain how you can find an analytical solution for $w_i \;\; i = 0, 1, \ldots, 5$.

To find an analytical solution for the coefficients (w_i) in the quadratic mapper function $(f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 (x_1)^2 + w_5 (x_2)^2)$, we use linear regression. Our goal is to minimize the sum of squared errors between the predicted values and the actual target values using the given training data.

Below is the deriving for the analytical solution:

1. We create a design matrix $(\mathbf{X})$ from the training data, where each row represents a sample $((x_1, x_2))$ and each column represents the features $(x_1, x_2, x_1 x_2, (x_1)^2, (x_2)^2)$ as follows:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{2,1} & x_{1,1}x_{2,1} & (x_{1,1})^2 & (x_{2,1})^2 \\ 1 & x_{1,2} & x_{2,2} & x_{1,2}x_{2,2} & (x_{1,2})^2 & (x_{2,2})^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{1,N} & x_{2,N} & x_{1,N}x_{2,N} & (x_{1,N})^2 & (x_{2,N})^2 \end{bmatrix}$$

1. We should use the substitioun method in order to solve the system.

Initially, we replace (x_1) by (z_1), (x_2) by (z_2), $(x_1 x_2)$ by (z_3), (x_1^2) by (z_4), and (x_2^2) by (z_5), and we rewrite the quadratic mapper function as follows:

$f(z_1, z_2, z_3, z_4, z_5) = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5$

Now, we want to find an analytical solution for the coefficients $(w_0, w_1, w_2, w_3, w_4, w_5)$ in terms of the new variables $(z_1, z_2, z_3, z_4, z_5)$ and the training data.

Let's denote the design matrix $(\mathbf{Z})$ with the new variables:

$$\mathbf{Z} = \begin{bmatrix} 1 & z_{1,1} & z_{2,1} & z_{3,1} & z_{4,1} & z_{5,1} \\ 1 & z_{1,2} & z_{2,2} & z_{3,2} & z_{4,2} & z_{5,2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & z_{1,N} & z_{2,N} & z_{3,N} & z_{4,N} & z_{5,N} \end{bmatrix}$$

1. We create a target vector $(\mathbf{t})$ that contains the corresponding target values $(t_1, t_2, \ldots, t_N)$.

$$\ \mathbf{t} =

$$\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

$$

1. We should define a weight vector $(\mathbf{w})$ that contains the coefficients $(w_0, w_1, w_2, w_3, w_4, w_5)$.

$$\ \mathbf{w} =

$$\begin{bmatrix} w_0, \\ w_1, \\ w_2, \\ w_3, \\ w_4, \\ w_5 \end{bmatrix}$$

$$

1. Now we can use linear regression to find the analytical solution for ($\mathbf{w}$). The solution is as bellow:

$$[\mathbf{w} = (\mathbf{Z}^T \mathbf{Z})^{-1} \mathbf{Z}^T \mathbf{t}]$$

Here's what each part of the equation represents:

- $((\mathbf{Z}^T\mathbf{Z}))$ is the matrix of inner products of the feature vectors.
- $((\mathbf{Z}^T\mathbf{Z})^{-1})$ is the inverse of this matrix.
- $(\mathbf{Z}^T)$ is the transpose of the design matrix $(\mathbf{Z})$.
- $(\mathbf{t})$ is the target vector.

Once we calculate $(\mathbf{w})$, we will have the analytical solution for $(w_0, w_1, w_2, w_3, w_4, w_5)$, and we can use these coefficients in our quadratic mapper function to make predictions for the new data points.

Question 4 (3 points)

In this problem, you will be working with the beer dataset with information about the foam height (in cm) from 3 brands of beer over 15 measurement times (in seconds) after the time of pour.

In [2]:

```
import pandas as pd

beer_data = pd.read_csv('/content/beer_foam.csv')

beer_data
```

Out[2]:

	Time	Erdinger	Augustinerbrau	Budweiser
0	0	17.0	14.0	14.0
1	15	16.1	11.8	12.1
2	30	14.9	10.5	10.9
3	45	14.0	9.3	10.0
4	60	13.2	8.5	9.3
5	75	12.5	7.7	8.6
6	90	11.9	7.1	8.0
7	105	11.2	6.5	7.5
8	120	10.7	6.0	7.0
9	150	9.7	5.3	6.2
10	180	8.9	4.4	5.5
11	210	8.3	3.5	4.5
12	240	7.5	2.9	3.5
13	300	6.3	1.3	2.0
14	360	5.2	0.7	0.9

Consider the first 12 samples as the training set, and the last 3 samples as the test set.

In [3]:
```
# Training and test sets
x_train = beer_data['Time'].to_numpy()[:12]
x_test  = beer_data['Time'].to_numpy()[12:]

# Training and test labels
t_train = beer_data.drop('Time', axis=1).iloc[:12]
t_test  = beer_data.drop('Time', axis=1).iloc[12:]
#t_train and t_test contain 3 target vectors.
```

Use the Python code implementation we built in class to help you train a mapper function of the form:

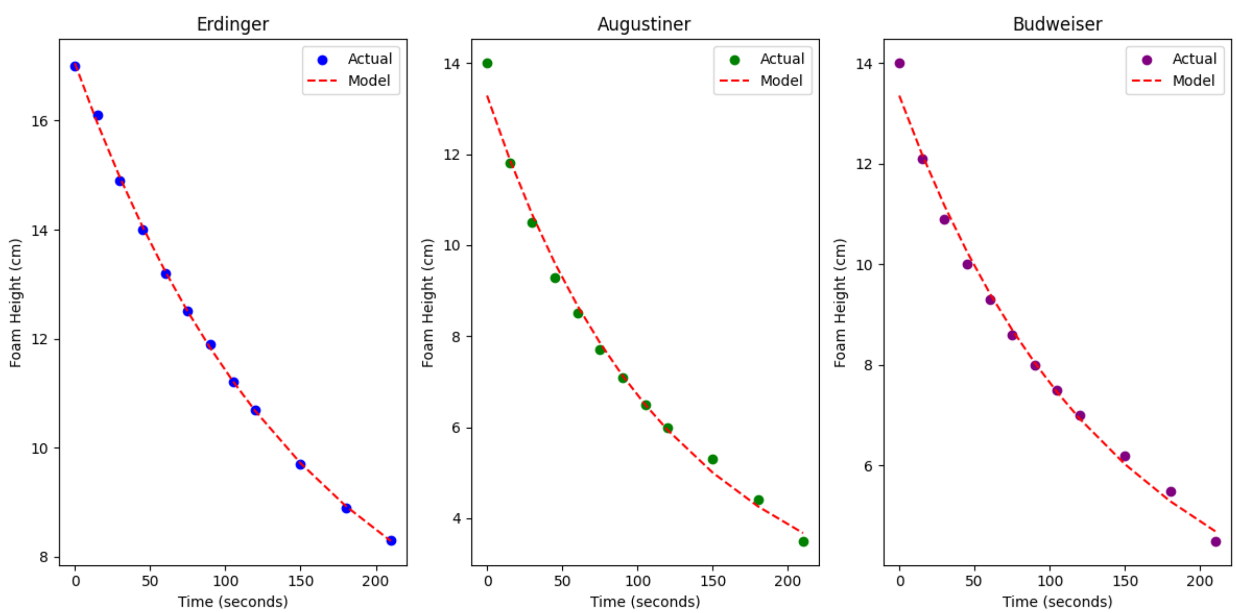
$$y(x) = \exp\left(\sum_{j=0}^{M} w_j x^j\right) = \exp(\mathbf{X}\mathbf{w})$$

Answer the following questions:

1. (1 point) **For each brand, train a mapper function with $M = 2$ using the training data. Plot the model prediction.**
2. (1 point) **Use each trained model to make predictions for the test data.**
3. (1 point) **For each brand, predict foam height at $t = 450$ seconds.**

In [8]:
```
# Import necessary libraries
import numpy as np  # Import NumPy for numerical operations
import matplotlib.pyplot as plt  # Import Matplotlib for plotting

# Training and test set
x_train = beer_data['Time'].to_numpy()[:12]  # Extracting the 'Time' column va
x_test = beer_data['Time'].to_numpy()[12:]  # Extract the 'Time' column values

# Training and test labels
t_train_erding = beer_data['Erdinger'].to_numpy()[:12]  # Extract the 'Erdinger
t_train_augustiner = beer_data['Augustinerbrau'].to_numpy()[:12]  # Extract the
t_train_budweiser = beer_data['Budweiser'].to_numpy()[:12]  # Extract the 'Budw
t_test = beer_data.drop('Time', axis=1).iloc[12:]  # Removing the 'Time' column

# Function to train a mapper with M=2
def train_mapper(x, t, M):
    X = np.column_stack([x ** i for i in range(M + 1)])  # Creating a designing
    w = np.linalg.lstsq(X, np.log(t), rcond=None)[0]  # Use the least squares t
    return w

# Function to predict using a trained mapper
def predict_mapper(x, w):
    X = np.column_stack([x ** i for i in range(len(w))])  # Create the same des
    predictions = np.exp(np.dot(X, w))  # Make predictions using the trained we
    return predictions
```

In [9]:
```
M = 2  # Setting the degree of the polynomial model to 2

# Training the mapper functions for each brand using the training data
w_erding = train_mapper(x_train, t_train_erding, M)  # Train the Erdinger branc
w_augustiner = train_mapper(x_train, t_train_augustiner, M)  # Train the August
w_budweiser = train_mapper(x_train, t_train_budweiser, M)  # Train the Budweise

# Plotting the model predictions for the training data
plt.figure(figsize=(12, 6))  # Create a figuere for plotting with an specific s

# Creating subplots for each brand's model prediction
plt.subplot(131)  # Creating the first subsplot
plt.scatter(x_train, t_train_erding, label='Actual', color='blue')  # Scatter p
plt.plot(x_train, predict_mapper(x_train, w_erding), label='Model', linestyle='
plt.title('Erdinger')  # Set the title for this subplot
plt.xlabel('Time (seconds)')  # Set the x-axis label
plt.ylabel('Foam Height (cm)')  # Set the y-axis label
plt.legend()  # Displays the legends

plt.subplot(132)  # Creating the second subplot
plt.scatter(x_train, t_train_augustiner, label='Actual', color='green')  # Scat
plt.plot(x_train, predict_mapper(x_train, w_augustiner), label='Model', linesty
plt.title('Augustiner')  # Setting the title for this subplot
plt.xlabel('Time (seconds)')  # Set the x-axis labels
plt.ylabel('Foam Height (cm)')  # Set the y-axis labels
plt.legend()  # Display the legend

plt.subplot(133)  # Create the third subplot
plt.scatter(x_train, t_train_budweiser, label='Actual', color='purple')  # Scat
plt.plot(x_train, predict_mapper(x_train, w_budweiser), label='Model', linestyl
plt.title('Budweiser')  # Seting the title for this subplot
plt.xlabel('Time (seconds)')  # Seting the x-axis label
plt.ylabel('Foam Height (cm)')  # Seting the y-axis label
plt.legend()  # Display the legend

plt.tight_layout()  # Adjust subplot layouts for better presentation
plt.show()  # Display the entire plot
```



In [10]:
```
# Using the trained Erdinger brand model to make predictions for the test datas
predictions_erding_test = predict_mapper(x_test, w_erding)

# Using the trained Augustiner brands model to make prediction for the test dat
predictions_augustiner_test = predict_mapper(x_test, w_augustiner)

# Use=ing the trained Budweiser brand model to make predictions for the test da
predictions_budweiser_test = predict_mapper(x_test, w_budweiser)

# Printing the predictions for the test datas for each brand
print("Predictions for test data:")
print("Erdinger:", predictions_erding_test)  # Print Erdinger brand predictions
print("Augustiner:", predictions_augustiner_test)  # Print Augustiner brand pre
print("Budweiser:", predictions_budweiser_test)  # Print Budweiser brand predic
```

```
Predictions for test data:
Erdinger: [7.75897384 7.00147073 6.56047876]
Augustiner: [3.20719818 2.52974218 2.09188487]
Budweiser: [4.20510132 3.47472968 2.98611627]
```

In [11]:
```
t_predict = 450  # Seting the time (t) for which we want to predict foam height

# Use the trained Erdinger brand model for predict foam height at t=450 seconds
foam_height_erding_predict = predict_mapper(np.array([t_predict]), w_erding)

# Use the trained Augustiner brand model for predict foam height at t=450 secon
foam_height_augustiner_predict = predict_mapper(np.array([t_predict]), w_august

# Use the trained Budweiser brand model for predict foam height at t=450 seconc
foam_height_budweiser_predict = predict_mapper(np.array([t_predict]), w_budweis

# Print the predicted foam height for each brand at t=450 seconds
print("\nPredicted foam height at t=450 seconds:")
print("Erdinger:", foam_height_erding_predict[0])  # Print Erdinger brand's pre
print("Augustiner:", foam_height_augustiner_predict[0])  # Print Augustiner bra
print("Budweiser:", foam_height_budweiser_predict[0])  # Print Budweiser brand
```

```
Predicted foam height at t=450 seconds:
Erdinger: 6.386049676706654
Augustiner: 1.718653297796165
Budweiser: 2.560574185787348
```

Question 5 (2.5 points)

Consider the [computer hardware dataset**]** **(**https://archive.ics.uci.edu/ml/datasets/Computer+Hardware**). The goal is to predict the estimated relative performance (ERP) of a CPU core as a function of 9 features (or independent variables):**

- **Vendor name:** 30 (adviser, amdahl,apollo, basf, bti, burroughs, c.r.d, cambex, cdc, dec, dg, formation, four-phase, gould, honeywell, hp, ibm, ipl, magnuson, microdata, nas, ncr, nixdorf, perkin-elmer, prime, siemens, sperry, sratus, wang)
- **Model Name:** many unique symbols

- **MYCT:** machine cycle time in nanoseconds (integer)
- **MMIN:** minimum main memory in kilobytes (integer)
- **MMAX:** maximum main memory in kilobytes (integer)
- **CACH:** cache memory in kilobytes (integer)
- **CHMIN:** minimum channels in units (integer)
- **CHMAX:** maximum channels in units (integer)
- **PRP:** published relative performance (integer)

And the target is:

- **ERP:** estimated relative performance from the original article (integer).

In [12]:
```
import pandas as pd

hardware=pd.read_csv('/content/machine.data',
                     names=['Vendor','Model Name','MYCT','MMIN','MMAX','CACH','CHMI
hardware
```

Out[12]:

	Vendor	Model Name	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP	ERP
0	adviser	32/60	125	256	6000	256	16	128	198	199
1	amdahl	470v/7	29	8000	32000	32	8	32	269	253
2	amdahl	470v/7a	29	8000	32000	32	8	32	220	253
3	amdahl	470v/7b	29	8000	32000	32	8	32	172	253
4	amdahl	470v/7c	29	8000	16000	32	8	16	132	132
...	...	...	...	...	...	...	...	...	...	...
204	sperry	80/8	124	1000	8000	0	1	8	42	37
205	sperry	90/80-model-3	98	1000	8000	32	2	8	46	50
206	sratus	32	125	2000	8000	0	2	14	52	41
207	wang	vs-100	480	512	8000	32	0	0	67	47
208	wang	vs-90	480	1000	4000	0	0	0	45	25

209 rows × 10 columns

In [15]:
```
# Feature matrix
data = hardware.drop('ERP', axis=1)
data
```

Out[15]:

	Vendor	Model Name	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP
0	adviser	32/60	125	256	6000	256	16	128	198
1	amdahl	470v/7	29	8000	32000	32	8	32	269
2	amdahl	470v/7a	29	8000	32000	32	8	32	220
3	amdahl	470v/7b	29	8000	32000	32	8	32	172
4	amdahl	470v/7c	29	8000	16000	32	8	16	132
...	...	...	...	...	...	...	...	...	...
204	sperry	80/8	124	1000	8000	0	1	8	42
205	sperry	90/80-model-3	98	1000	8000	32	2	8	46
206	sratus	32	125	2000	8000	0	2	14	52
207	wang	vs-100	480	512	8000	32	0	0	67
208	wang	vs-90	480	1000	4000	0	0	0	45

209 rows × 9 columns

In [16]:

```
# Targeting labels
target = hardware['ERP']
target
```

Out[16]:

```
0      199
1      253
2      253
3      253
4      132
      ...
204     37
205     50
206     41
207     47
208     25
Name: ERP, Length: 209, dtype: int64
```

Consider only the numerical features: MYCT, MMIN, MMAX, CACH, CHMIN, CHMAX, and PRP.

Answer the following questions:

1. (0.5 points) **Partition the data randomly using an 80/20 split. See sklearn.model_selection.train_test_split.**
2. (1 point) **Train a multivariate regression model using the training data.**
3. (1 point) **Make predictions in the test set and report performance.**

In [17]:

```
# Import necessary libraries and modules for machine learning and regression ar

# Import the train_test_split function from scikit-learn for splitting data int
from sklearn.model_selection import train_test_split
```

```
# Import the LinearRegression class from scikit-learn for linear regression moc
from sklearn.linear_model import LinearRegression

# Import mean_squared_error and r2_score functions from scikit-learn for regres
from sklearn.metrics import mean_squared_error, r2_score
```

In [18]:
```
# Defineing a list of numerical features to be used as input features
numerical_features = ['MYCT', 'MMIN', 'MMAX', 'CACH', 'CHMIN', 'CHMAX', 'PRP']

# Defining the target variable you want to estimate
estimate = ['ERP']

# Extracting the input features (X) from the 'data' DataFrame using the selecte
X = data[numerical_features]

# Extract the target variable (y) from the 'target' data (assuming 'target' is
y = target

# Split the data into training and testing sets using the train_test_split func
# - X_train: The training data for input features
# - X_test: The testing data for input features
# - y_train: The training data for the target variable
# - y_test: The testing data for the target variable
# - test_size: The proportion of the data to be used for testing (here, 20%)
# - random_state: A seed for the random number generator to ensure reproducibil
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randon
```

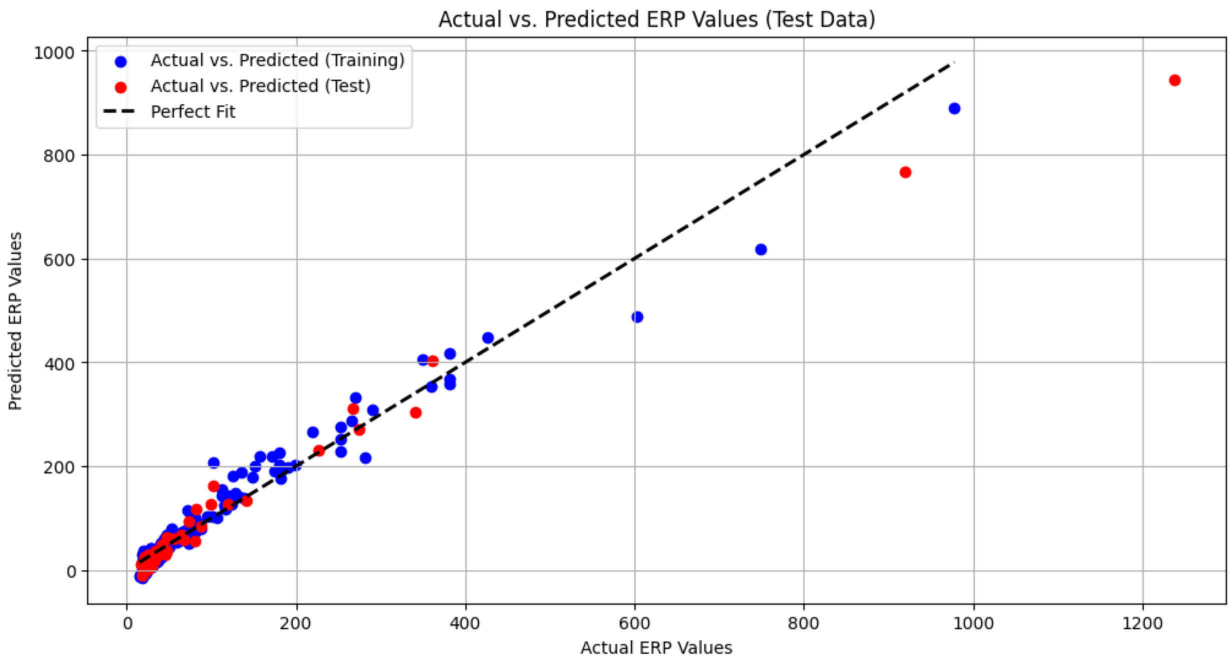
In [19]:
```
# 2. Training a multivariate regression model with the training data
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[19]:
```
▾ LinearRegression
LinearRegression()
```

In [20]:
```
# Useing the trained 'model' to make predictions on the test data 'X_test'
y_pred = model.predict(X_test)

# Calculate performance metrics - Mean Squared Error (MSE) and R-squared (R2)
# - 'mean_squared_error' computes the MSE between the true 'y_test' values and
# - 'r2_score' computes the R-squared (R2) score between 'y_test' and 'y_pred'
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Printing the calculated performance metrics
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")

# Importing the matplotlib library for the plotting
import matplotlib.pyplot as plt

# Creating a figure for the ploting
plt.figure(figsize=(12, 6))

# Scattering the plot of actual vs predicted ERP values on the training data (i
plt.scatter(y_train, model.predict(X_train), color='blue', label='Actual vs. Pr
plt.xlabel('Actual ERP Values')
```

```
plt.ylabel('Predicted ERP Values')
plt.title('Actual vs. Predicted ERP Values (Training Data)')
plt.grid(True)

# Scattering the plot of actual vs predicted ERP values on the test data (in re
plt.scatter(y_test, y_pred, color='red', label='Actual vs. Predicted (Test)')
plt.xlabel('Actual ERP Values')
plt.ylabel('Predicted ERP Values')
plt.title('Actual vs. Predicted ERP Values (Test Data)')

# Ploting a perfect fit line for reference (in black dashed line)
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)], linestyle=

# Displaying the legend to differentiate between training and test data points

plt.legend()

# Showing the plot
plt.show()
```

```
Mean Squared Error (MSE): 3007.8898321639304
R-squared (R2): 0.9440465034138787
```



Mean square error (MSE): 3007.8898

The MSE is a measure of the difference between actual and predicted values. In this case, the MSE would be 3007.8898, which represents the error between the actual and predicted ERP values of the experimental data

R-score (R^2): 0.9440

The R-squared (R^2) score measures the predictability of objective variables (ERP) variables from input characteristics. The R^2 score of 0.9440 indicates that about 94.40% of the variance in ERP can be explained by the model. This is a good indicator of model performance, as (R^2) values close to 1 indicate good predictive power. Overall, the model appears to perform well at low MSE and high (R^2) scores, indicating that it can predict ERP values well on test data

---

On-Time (1 point)

Submit your assignment before the deadline.

Submit Your Solution

Confirm that you've successfully completed the assignment.

Along with the Notebook, include a PDF of the notebook with your solutions.

`add` and `commit` the final version of your work, and `push` your code to your GitHub repository.

Submit the URL of your GitHub Repository as your assignment submission on Canvas.
