```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# --- 1. Data Loading and Preparation ---

# Create a sample DataFrame for demonstration purposes
# Replace this section with loading your actual data
data = {
    'batsman': ['PlayerA', 'PlayerB', 'PlayerA', 'PlayerC', 'PlayerB', 'PlayerA'],
    'bowler': ['BowlerX', 'BowlerY', 'BowlerX', 'BowlerZ', 'BowlerY', 'BowlerZ'],
    'over': [1, 2, 3, 1, 2, 3],
    'team_total_runs': [5, 15, 25, 6, 18, 30],
    'extra_score': [0, 1, 0, 0, 0, 1],
    'runs_scored': [6, 8, 10, 7, 9, 11] # This is the target variable
}
df = pd.DataFrame(data)

print("Sample DataFrame created:")
print(df.head())

# --- 2. Handling Categorical Variables ---

# One-hot encode the categorical variables
categorical_cols = ['batsman', 'bowler']
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

print("\nDataFrame after one-hot encoding:")
print(df.head())
print(f"Columns after encoding: {df.columns.tolist()}")

# --- 3. Defining Features and Target Variable ---

# Define your features (X) and target (y)
target_column = 'runs_scored'
if target_column in df.columns:
    X = df.drop(target_column, axis=1)
    y = df[target_column]
    print(f"\nFeatures (X) and target (y) defined. Target: '{target_column}'")
else:
    print(f"\nError: Target column '{target_column}' not found in the DataFrame.")
    # Exit if the target column is not found
    exit()

# Ensure that X and y have the same number of rows
if X.shape[0] != y.shape[0]:
    print("Error: Number of rows in features (X) and target (y) do not match.")
    exit()

# --- 4. Splitting Data for Training and Testing ---

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(f"\nData split into training (80%) and testing (20%) sets.")
print(f"Training set shape: X_train - {X_train.shape}, y_train - {y_train.shape}")
print(f"Testing set shape: X_test - {X_test.shape}, y_test - {y_test.shape}")

# --- 5. Building and Training the Linear Regression Model ---

# Create a Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
print("\nLinear Regression model trained successfully.")

# --- 6. Evaluating the Model ---

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
```

```
rmse = np.sqrt(mse) # Root Mean Squared Error
r2 = r2_score(y_test, y_pred)

print("\n--- Model Evaluation ---")
print(f'Mean Absolute Error (MAE): {mae:.2f}')
print(f'Mean Squared Error (MSE): {mse:.2f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
print(f'R-squared (R2): {r2:.2f}')

# --- 7. Making Predictions (Example) ---

# Example of making a prediction for new data
# To make a prediction for new data, you need to create a new DataFrame
# with the same columns as your training data (X_train), including
# the one-hot encoded columns.

# Create a sample new data point for demonstration
# This must match the structure of X_train after encoding
# Let's create a new data point where PlayerA is batting against BowlerY in over 4
# with team total runs 40 and extra score 2.

# Create a dictionary with the new data
new_data_dict = {
    'over': [4],
    'team_total_runs': [40],
    'extra_score': [2],
    # Include all dummy columns present in X_train, setting values to 0 or 1
    'batsman_PlayerB': [0], # Assuming PlayerA is batting, so PlayerB and PlayerC are 0
    'batsman_PlayerC': [0],
    'bowler_BowlerY': [1], # Assuming BowlerY is bowling, so BowlerX and BowlerZ are 0
    'bowler_BowlerZ': [0]
}

# Create a DataFrame for the new data point
new_data = pd.DataFrame(new_data_dict)

# Ensure the new data has the exact same columns as X_train and in the same order
# This is a crucial step for prediction. We need to align the columns.
# Get the columns from the training data
train_cols = X_train.columns

# Reindex the new data DataFrame to match the training data columns
new_data = new_data.reindex(columns=train_cols, fill_value=0)


try:
    predicted_runs = model.predict(new_data)
    print(f'\nPredicted runs for new data: {predicted_runs[0]:.2f}')
except Exception as e:
    print(f"\nError making prediction on new data: {e}")
```

```
Sample DataFrame created:
   batsman   bowler  over  team_total_runs  extra_score  runs_scored
0  PlayerA  BowlerX     1                5            0            6
1  PlayerB  BowlerY     2               15            1            8
2  PlayerA  BowlerX     3               25            0           10
3  PlayerC  BowlerZ     1                6            0            7
4  PlayerB  BowlerY     2               18            0            9

DataFrame after one-hot encoding:
   over  team_total_runs  extra_score  runs_scored  batsman_PlayerB  \
0     1                5            0            6            False
1     2               15            1            8             True
2     3               25            0           10            False
3     1                6            0            7            False
4     2               18            0            9             True

   batsman_PlayerC  bowler_BowlerY  bowler_BowlerZ
0            False           False           False
1            False            True           False
2            False           False           False
3             True           False            True
4            False            True           False
Columns after encoding: ['over', 'team_total_runs', 'extra_score', 'runs_scored', 'batsman_PlayerB', 'batsman_PlayerC', 'bowler_BowlerY'

Features (X) and target (y) defined. Target: 'runs_scored'

Data split into training (80%) and testing (20%) sets.
Training set shape: X_train - (4, 7), y_train - (4,)
```

```
Testing set shape: X_test - (2, 7), y_test - (2,)

Linear Regression model trained successfully.

--- Model Evaluation ---
Mean Absolute Error (MAE): 0.67
Mean Squared Error (MSE): 0.46
Root Mean Squared Error (RMSE): 0.68
R-squared (R2): 0.54

Predicted runs for new data: 12.69
```

```
# After training the model:
# model.fit(X_train, y_train)

print("\n--- Model Coefficients ---")

# Get the feature names
feature_names = X_train.columns

# Get the coefficients
coefficients = model.coef_

# Create a dictionary to store feature names and their coefficients
coef_dict = dict(zip(feature_names, coefficients))

# Print the coefficients
for feature, coef in coef_dict.items():
    print(f'{feature}: {coef:.4f}')
```

```
--- Model Coefficients ---
over: -0.0446
team_total_runs: 0.1664
extra_score: 0.0879
batsman_PlayerB: 0.0601
batsman_PlayerC: -0.0078
bowler_BowlerY: 0.0601
bowler_BowlerZ: 0.0801
```

```
# Assuming 'coef_dict' is populated from the previous step

batsman_coefs = {k: v for k, v in coef_dict.items() if k.startswith('batsman_')}
bowler_coefs = {k: v for k, v in coef_dict.items() if k.startswith('bowler_')}

# Add the baseline batsman (if applicable) with a coefficient of 0
# You need to know which batsman was dropped (the one not present in batsman_coefs keys)
# For example, if 'batsman_PlayerA' was dropped:
# batsman_coefs['batsman_PlayerA'] = 0

# Sort batsmen by coefficient (higher means potentially "better" for scoring runs)
sorted_batsmen = sorted(batsman_coefs.items(), key=lambda item: item[1], reverse=True)

# Add the baseline bowler (if applicable) with a coefficient of 0
# You need to know which bowler was dropped
# For example, if 'bowler_BowlerX' was dropped:
# bowler_coefs['bowler_BowlerX'] = 0

# Sort bowlers by coefficient (more negative/less positive means potentially "better" for conceding fewer runs)
sorted_bowlers = sorted(bowler_coefs.items(), key=lambda item: item[1])

print("\nSorted Batsmen by Coefficient (Higher means more runs):")
for batsman, coef in sorted_batsmen:
    print(f'{batsman}: {coef:.4f}')

print("\nSorted Bowlers by Coefficient (Lower means fewer runs conceded):")
for bowler, coef in sorted_bowlers:
    print(f'{bowler}: {coef:.4f}')
```

```
Sorted Batsmen by Coefficient (Higher means more runs):
batsman_PlayerB: 0.0601
batsman_PlayerC: -0.0078

Sorted Bowlers by Coefficient (Lower means fewer runs conceded):
bowler_BowlerY: 0.0601
bowler_BowlerZ: 0.0801
```

Start coding or generate with AI.