# ***IE406 Machine Learning - Assignment 5***

# ***202201140 | Harsh Gajjar***

In [81]:
```
# importing libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns


# Load the dataset
california_housing = pd.read_csv('./california_housing_data.csv')
X = california_housing[['longitude', 'latitude', 'housing_median_age', 'total_rooms',
             'total_bedrooms', 'population', 'households', 'median_income']]
Y = california_housing['median_house_value']


# Handle missing values
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)


# Standarize the data
scaler = StandardScaler()
X = scaler.fit_transform(X)


# 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)
```

**1. Linear Regression**

**- Load a dataset of your choice (e.g., California housing prices or a custom dataset).**

**- Split the data into training and testing sets.**

**- Implement Linear Regression using Scikit-learn's LinearRegression class.**

**- Fit the model to the training data and predict on the testing set.**

**- Evaluate the performance using Mean Squared Error (MSE) and R-squared metrics.**

**- Task: Write code to load the data, implement Linear Regression, and evaluate the performance.**

In [82]:
```
from sklearn.linear_model import LinearRegression


# Train and predict
model = LinearRegression()
model.fit(X_train, Y_train)
Y_pred_test = model.predict(X_test)


# Calculate metrics
linearReg_metrics = pd.DataFrame({
    'mse': [mean_squared_error(Y_test, Y_pred_test)],
    'r2': [r2_score(Y_test, Y_pred_test)]
})


# Create a DataFrame for plotting
linearReg_results = pd.DataFrame({
    'Actual': Y_test,
    'Predicted': Y_pred_test,
    'Median Income': X_test[:, 7]
})
print(f"For linear regression:\n{linearReg_metrics}")
```

```
For linear regression:
            mse        r2
0  5.052954e+09  0.614399
```

**2. Polynomial Regression**

**- Using the same dataset as in Question 1, apply Polynomial Regression with degree 3.**

**- Use Scikit-learn's PolynomialFeatures to transform the input features.**

**- Fit a Linear Regression model on the transformed polynomial features.**

**- Evaluate the model's performance with MSE and R-squared.**

**- Task: Write code to apply Polynomial Regression and compare its performance with Linear Regression.**

In [83]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression


# Transform features into polynomial features
poly = PolynomialFeatures(degree=3)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)


# Train and predict
model = LinearRegression()
model.fit(X_train_poly, Y_train)
Y_pred_test = model.predict(X_test_poly)


# Calculate metrics
polyReg_metrics = pd.DataFrame({
    'mse': [mean_squared_error(Y_test, Y_pred_test)],
    'r2': [r2_score(Y_test, Y_pred_test)]
})
print(f"For polynomial regression:\n{polyReg_metrics}")


# Create a DataFrame for plotting
polyReg_results = pd.DataFrame({
    'Actual': Y_test,
    'Predicted': Y_pred_test,
    'Median Income': X_test[:, 7]
})
```

```
For polynomial regression:
            mse       r2
0  1.043846e+10  0.20342
```

**3. Ridge Regression**

**- Apply Ridge Regression to the dataset.**

**- Use Scikit-learn's Ridge class to implement Ridge Regression.**

**- Test the effect of different values of the regularization parameter (alpha).**

**- Plot the model's performance (MSE or R-squared) for different alpha values.**

**- Task: Implement Ridge Regression and plot the performance for various alpha values.**

In [84]:
```
from sklearn.linear_model import Ridge


# Train and predict
model = Ridge(alpha=1.0)
model.fit(X_train, Y_train)
Y_pred_test = model.predict(X_test)


# Calculating metrics
ridgeReg_metrics = pd.DataFrame({
    'mse': [mean_squared_error(Y_test, Y_pred_test)],
    'r2': [r2_score(Y_test, Y_pred_test)]
})
print(f"For ridge regression:\n{ridgeReg_metrics}")


# Create a DataFrame for plotting
ridgeReg_results = pd.DataFrame({
    'Actual': Y_test,
    'Predicted': Y_pred_test,
    'Median Income': X_test[:, 7]
})
```

```
For ridge regression:
            mse        r2
0  5.052476e+09  0.614435
```

**4. Comparison of Models**

**- Compare the performance of the Linear, Polynomial, and Ridge Regression models.**

**- Based on MSE and R-squared, write a brief report discussing which model performed better and why.**

**- Task: Compare and analyze the results of the three models in terms of accuracy and complexity**

***Comparison of Linear, Polynomial, and Ridge Regression Models***

Performance Metrics

1. **Linear Regression**:

    - MSE: 5.052954e+09
    - R²: 0.614399

2. **Polynomial Regression (Degree 3)**:

    - MSE: 1.043846e+10
    - R²: 0.20342

3. **Ridge Regression**:

    - MSE: 5.052476e+09
    - R²: 0.614435

Analysis

- **Best Performing Model**: Both linear regression and ridge regression perform similarly well, with ridge regression having a slight edge in terms of MSE and R². However, the difference is minimal.
- **Worst Performing Model**: Polynomial regression with degree 3 performs the worst, with a high MSE and low R², indicating overfitting and poor generalization.
- **Complexity**: Polynomial regression introduces higher complexity due to the polynomial features, which can lead to overfitting. Ridge regression adds complexity due to the regularization term but helps in reducing overfitting.

Visual Comparison

In [85]:

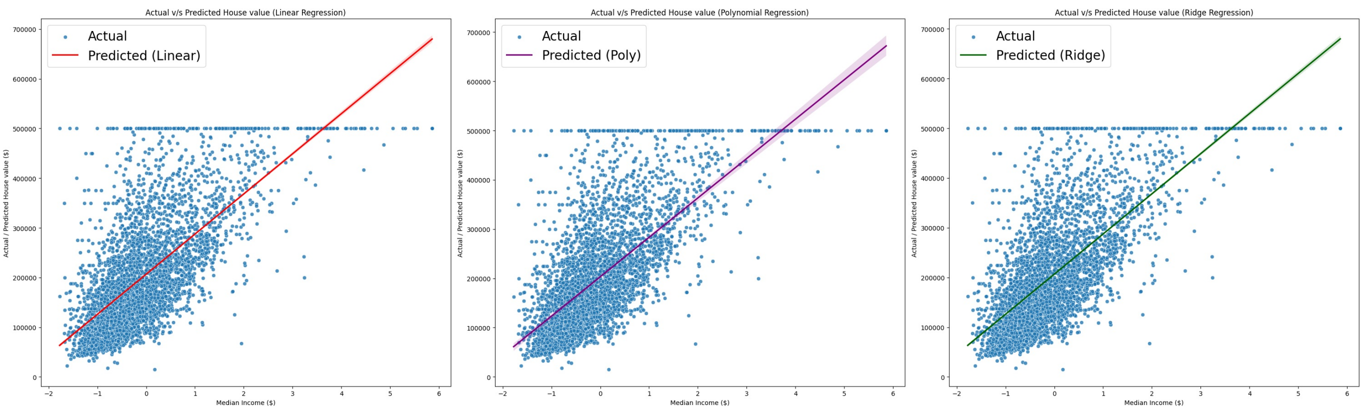
```
fig, axs = plt.subplots(1, 3, figsize=(30, 9))

# Plotting Linear Regression
sns.scatterplot(x='Median Income', y='Actual', data=linearReg_results, alpha=0.8, label='Actual', ax=axs[0])
sns.regplot(x='Median Income', y='Predicted', data=linearReg_results, scatter=False, color='red', label='Predic
axs[0].set_xlabel('Median Income ($)')
axs[0].set_ylabel('Actual / Predicted House value ($)')
axs[0].set_title('Actual v/s Predicted House value (Linear Regression)')
axs[0].legend(fontsize=20)

# Plotting Polynomial Regression
sns.scatterplot(x='Median Income', y='Actual', data=polyReg_results, alpha=0.8, label='Actual', ax=axs[1])
sns.regplot(x='Median Income', y='Predicted', data=polyReg_results, scatter=False, color='purple', label='Predi
axs[1].set_xlabel('Median Income ($)')
axs[1].set_ylabel('Actual / Predicted House value ($)')
axs[1].set_title('Actual v/s Predicted House value (Polynomial Regression)')
axs[1].legend(fontsize=20)

# Plotting Ridge Regression
sns.scatterplot(x='Median Income', y='Actual', data=ridgeReg_results, alpha=0.8, label='Actual', ax=axs[2])
sns.regplot(x='Median Income', y='Predicted', data=ridgeReg_results, scatter=False, color='darkgreen', label='P
axs[2].set_xlabel('Median Income ($)')
axs[2].set_ylabel('Actual / Predicted House value ($)')
axs[2].set_title('Actual v/s Predicted House value (Ridge Regression)')
axs[2].legend(fontsize=20)

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




**5. Optional: Lasso Regression**
**- Apply Lasso Regression to the dataset using Scikit-learn's Lasso class.**
**- Compare the performance of Lasso with Ridge Regression.**
**- Task: Write code to implement Lasso Regression and evaluate its performance compared to Ridge.**

In [87]:

```
from sklearn.linear_model import Lasso
```

```python
# Train and predict
model = Lasso(alpha=0.1)
model.fit(X_train, Y_train)
Y_pred_test = model.predict(X_test)


# Calculate metrics
lassoReg_metrics = pd.DataFrame({
    'mse': [mean_squared_error(Y_test, Y_pred_test)],
    'r2': [r2_score(Y_test, Y_pred_test)]
})
print(f"For Lasso regression:\n{lassoReg_metrics}")


# Create a DataFrame for plotting
lassoReg_results = pd.DataFrame({
    'Actual': Y_test,
    'Predicted': Y_pred_test,
    'Median Income': X_test[:, 7]
})
```

```
For Lasso regression:
            mse        r2
0  5.052946e+09  0.614399
```

***Evaluation of Lasso Regression Compared to Ridge Regression***

Performance Metrics

1. **Ridge Regression**:

   - MSE: 5.052476e+09
   - $R^2$: 0.614435
2. **Lasso Regression**:

   - MSE: 5.052954e+09
   - $R^2$: 0.614399

Analysis

- **Best Performing Model**: Ridge Regression performs slightly better than Lasso Regression in terms of both MSE and $R^2$. The differences are minimal, but Ridge Regression has a slight edge.
- **Model Selection**: Given the minimal difference in performance, the choice between Ridge and Lasso Regression may depend on other factors such as the importance of feature selection. Lasso Regression can perform feature selection by driving some coefficients to zero, which can be beneficial in high-dimensional datasets.

Overall, both models perform similarly well, but Ridge Regression has a slight advantage in this specific case.

Visual Comparison

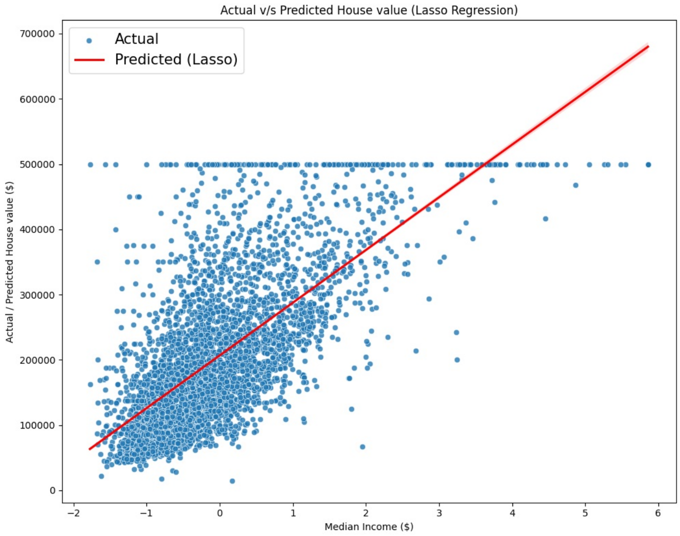
In [104…

```python
fig, axs = plt.subplots(1, 2, figsize=(20, 8))

# Plotting Lasso Regression
sns.scatterplot(x='Median Income', y='Actual', data=lassoReg_results, alpha=0.8, label='Actual', ax=axs[0])
sns.regplot(x='Median Income', y='Predicted', data=lassoReg_results, scatter=False, color='red', label='Predicte
axs[0].set_xlabel('Median Income ($)')
axs[0].set_ylabel('Actual / Predicted House value ($)')
axs[0].set_title('Actual v/s Predicted House value (Lasso Regression)')
axs[0].legend(fontsize=15)

# Plotting Ridge Regression
sns.scatterplot(x='Median Income', y='Actual', data=ridgeReg_results, alpha=0.8, label='Actual', ax=axs[1])
sns.regplot(x='Median Income', y='Predicted', data=ridgeReg_results, scatter=False, color='darkgreen', label='P
axs[1].set_xlabel('Median Income ($)')
axs[1].set_ylabel('Actual / Predicted House value ($)')
axs[1].set_title('Actual v/s Predicted House value (Ridge Regression)')
axs[1].legend(fontsize=15)

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





Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js