Code

April 3, 2024

[ ]:
```
pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)

317.0/317.0 MB 1.9 MB/s eta 0:00:00m eta
0:00:01[36m0:00:04
  Preparing metadata (setup.py) … done
Collecting py4j==0.10.9.7 (from pyspark)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl.metadata (1.5 kB)
Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)

200.5/200.5 kB 15.6 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) … /
```

[8]:
```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.feature import StringIndexer, VectorAssembler,␣
 ↪PolynomialExpansion
from pyspark.ml.regression import LinearRegression, RandomForestRegressor,␣
 ↪DecisionTreeRegressor
from pyspark.ml.evaluation import RegressionEvaluator as RE
from pyspark.sql import SparkSession as SS
import matplotlib.pyplot as mpt
```

[9]:
```
# Create a SparkSession
sparkSession = SS.builder.appName("HousePriceAnalysis").getOrCreate()
```

[10]:
```
# Dataset Description
record_path = "/home/sujan-tumbaraguddi/Downloads/HPP/train.csv"
#https://www.kaggle.com/datasets/anmolkumar/house-price-prediction-challenge?
 ↪select=train.csv
record_name = "House Price Prediction Challenge"

# Load the dataset into a PySpark DataFrame
record = spark.read.csv(record_path, header=True, inferSchema=True)

# Explore the dataset
```

```
print(f"Dataset Name: {record_name}")
print(f"Number of rows: {record.count()}")
print(f"Number of columns: {len(record.columns)}")
record.printSchema()
record.show(5)
```

```
Dataset Name: House Price Prediction Challenge
Number of rows: 29451
Number of columns: 12
root
 |-- POSTED_BY: string (nullable = true)
 |-- UNDER_CONSTRUCTION: integer (nullable = true)
 |-- RERA: integer (nullable = true)
 |-- BHK_NO.: integer (nullable = true)
 |-- BHK_OR_RK: string (nullable = true)
 |-- SQUARE_FT: double (nullable = true)
 |-- READY_TO_MOVE: integer (nullable = true)
 |-- RESALE: integer (nullable = true)
 |-- ADDRESS: string (nullable = true)
 |-- LONGITUDE: double (nullable = true)
 |-- LATITUDE: double (nullable = true)
 |-- TARGET(PRICE_IN_LACS): double (nullable = true)

+---------+-----------------+----+-------+---------+-----------------+---------
----+------+--------------------+---------+---------+---------------------+
|POSTED_BY|UNDER_CONSTRUCTION|RERA|BHK_NO.|BHK_OR_RK|
SQUARE_FT|READY_TO_MOVE|RESALE|             ADDRESS|LONGITUDE|
LATITUDE|TARGET(PRICE_IN_LACS)|
+---------+-----------------+----+-------+---------+-----------------+---------
----+------+--------------------+---------+---------+---------------------+
|    Owner|                0|   0|      2|      BHK|      1300.236407|
1|     1|Ksfc Layout,Banga…| 12.96991| 77.59796|                 55.0|
|   Dealer|                0|   0|      2|      BHK|           1275.0|
1|     1|Vishweshwara Naga…|12.274538|76.644605|                 51.0|
|    Owner|                0|   0|      2|      BHK|933.1597222000001|
1|     1|    Jigani,Bangalore|12.778033|77.632191|                 43.0|
|    Owner|                0|   1|      2|      BHK|929.9211427000001|
1|     1|Sector-1 Vaishali…|  28.6423|  77.3445|                 62.5|
|   Dealer|                1|   0|      2|      BHK|999.0092470000001|
0|     1|    New Town,Kolkata|  22.5922|88.484911|                 60.5|
+---------+-----------------+----+-------+---------+-----------------+---------
----+------+--------------------+---------+---------+---------------------+
only showing top 5 rows
```

[11]:
```
# Handle missing values and drop unnecessary columns
record = record.drop("ADDRESS", "BHK_OR_RK")
```

```
record = record.withColumnRenamed("BHK_NO.", "BHK_NO")
```

[12]:
```
# Identify numerical columns
num_cols = [item[0] for item in record.dtypes if item[1].startswith('int') or␣
 ↪item[1].startswith('double')]

# Convert string columns to numeric using StringIndexer for categorical columns␣
 ↪with fewer unique values
string_cols = [col for col in record.columns if col not in num_cols]
indexers = [StringIndexer(inputCol=col, outputCol=f"{col}_index").fit(record)␣
 ↪for col in string_cols if
            record.select(col).distinct().count() <= 100]  # Adjust the␣
 ↪threshold as needed
for indexer in indexers:
    record = indexer.transform(record)

# Assemble features into a vector
feature_cols = [col + "_index" for col in string_cols if record.select(col).
 ↪distinct().count() <= 100] + num_cols  # Exclude the target column
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
record = assembler.transform(record)
```

[13]:
```
# Split the dataset into training and testing sets
Train_x, Test_y = record.randomSplit([0.8, 0.2], seed=42)
```

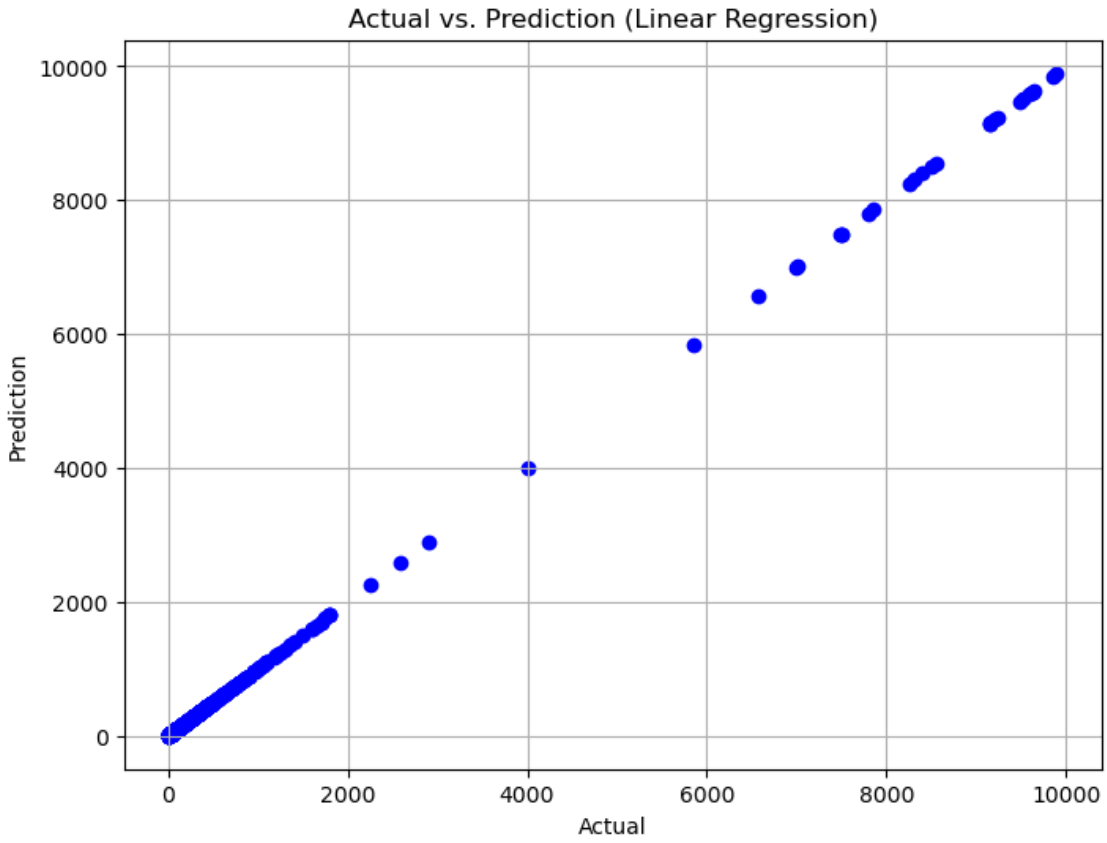
[14]:
```
# Define function to calculate MAE, RMSE, and R-squared
def evaluate_model(predictions, label_col):
    evaluatorRMSE = RE(labelCol=label_col, predictionCol="prediction",␣
 ↪metricName="rmse")
    evaluatorMAE = RE(labelCol=label_col, predictionCol="prediction",␣
 ↪metricName="mae")
    evaluatorR2 = RE(labelCol=label_col, predictionCol="prediction",␣
 ↪metricName="r2")
    rmse = evaluatorRMSE.evaluate(predictions)
    mae = evaluatorMAE.evaluate(predictions)
    r2 = evaluatorR2.evaluate(predictions)
    return rmse, mae, r2
```

[15]:
```
# Perform linear regression analysis
LR = LinearRegression(featuresCol="features", labelCol=num_cols[8])
lrModel = LR.fit(Train_x)
lrPredictions = lrModel.transform(Test_y)

# Evaluate the linear regression model
lrRMSE, lrMAE, lrR2 = evaluate_model(lrPredictions, num_cols[8])
print(f"Linear Regression RMSE: {lrRMSE}")
print(f"Linear Regression MAE: {lrMAE}")
```

```
print(f"Linear Regression R-squared: {lrR2}")

# Plot predictions vs. actual values for Linear Regression
predictionsLR = lrPredictions.select("prediction").rdd.map(lambda row: row[0]).
  ↪collect()
actual_valuesLR = lrPredictions.select(num_cols[8]).rdd.map(lambda row: row[0]).
  ↪collect()

mpt.figure(figsize=(8, 6))
mpt.scatter(actual_valuesLR, predictionsLR, color='blue')
mpt.title("Actual vs. Prediction (Linear Regression)")
mpt.xlabel("Actual")
mpt.ylabel("Prediction")
mpt.grid(True)
mpt.show()
```

```
24/04/03 04:23:05 WARN Instrumentation: [67470b73] regParam is zero, which might
cause numerical instability and overfitting.
24/04/03 04:23:07 WARN Instrumentation: [67470b73] Cholesky solver failed due to
singular covariance matrix. Retrying with Quasi-Newton solver.
```

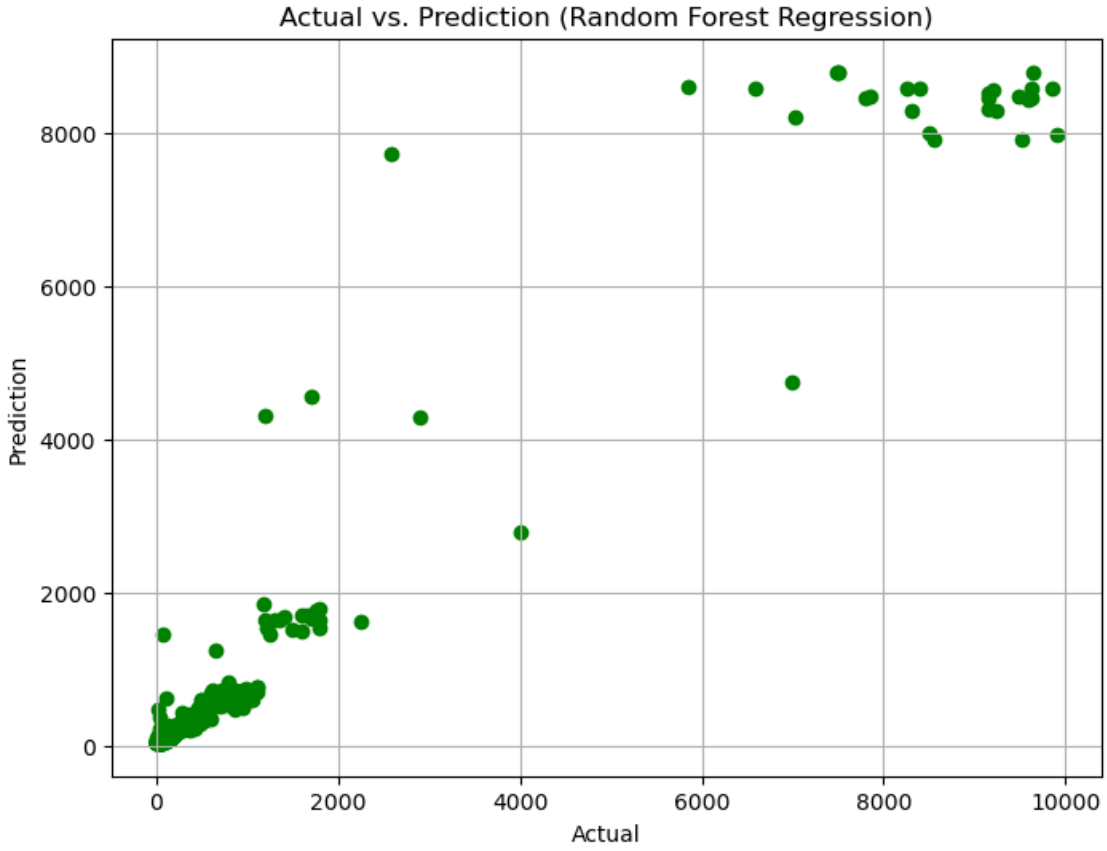
```
Linear Regression RMSE: 4.3661313407514455e-06
Linear Regression MAE: 3.2560721944423238e-06
Linear Regression R-squared: 1.0
```



[16]:
```
# Random Forest Regression
RF = RandomForestRegressor(featuresCol="features", labelCol=num_cols[8],␣
 ↪maxBins=100)  # Set maxBins to a smaller value
rfModel = RF.fit(Train_x)
rfPredictions = rfModel.transform(Test_y)

# Evaluate the random forest regression model
rfRMSE, rfMAE, rfR2 = evaluate_model(rfPredictions, num_cols[8])
print(f"Random Forest Regression RMSE: {rfRMSE}")
print(f"Random Forest Regression MAE: {rfMAE}")
print(f"Random Forest Regression R-squared: {rfR2}")

# Plot predictions vs. actual values for Random Forest Regression
predictionsRF = rfPredictions.select("prediction").rdd.map(lambda row: row[0]).
 ↪collect()
actual_valuesRF = rfPredictions.select(num_cols[8]).rdd.map(lambda row: row[0]).
 ↪collect()

mpt.figure(figsize=(8, 6))
mpt.scatter(actual_valuesRF, predictionsRF, color='green')
```

```
mpt.title("Actual vs. Prediction (Random Forest Regression)")
mpt.xlabel("Actual")
mpt.ylabel("Prediction")
mpt.grid(True)
mpt.show()
```

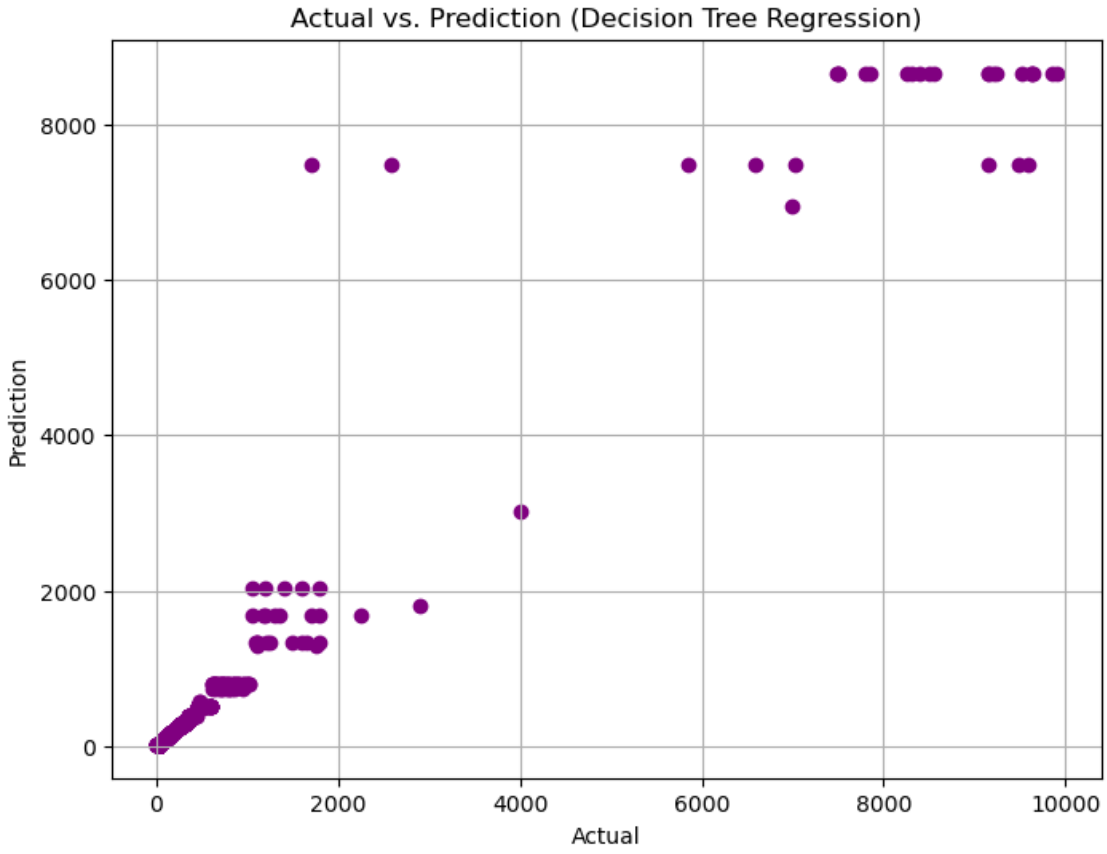
```
Random Forest Regression RMSE: 131.23740556168354
Random Forest Regression MAE: 30.973432462535538
Random Forest Regression R-squared: 0.9512972316838876
```



[17]:
```
# Decision Tree Regression
DT = DecisionTreeRegressor(featuresCol="features", labelCol=num_cols[8],␣
 ↪maxBins=100)  # Set maxBins to a smaller value
dtModel = DT.fit(Train_x)
dtPredictions = dtModel.transform(Test_y)

# Evaluate the decision tree regression model
dtRMSE, dtMAE, dtR2 = evaluate_model(dtPredictions, num_cols[8])
print(f"Decision Tree Regression RMSE: {dtRMSE}")
print(f"Decision Tree Regression MAE: {dtMAE}")
```

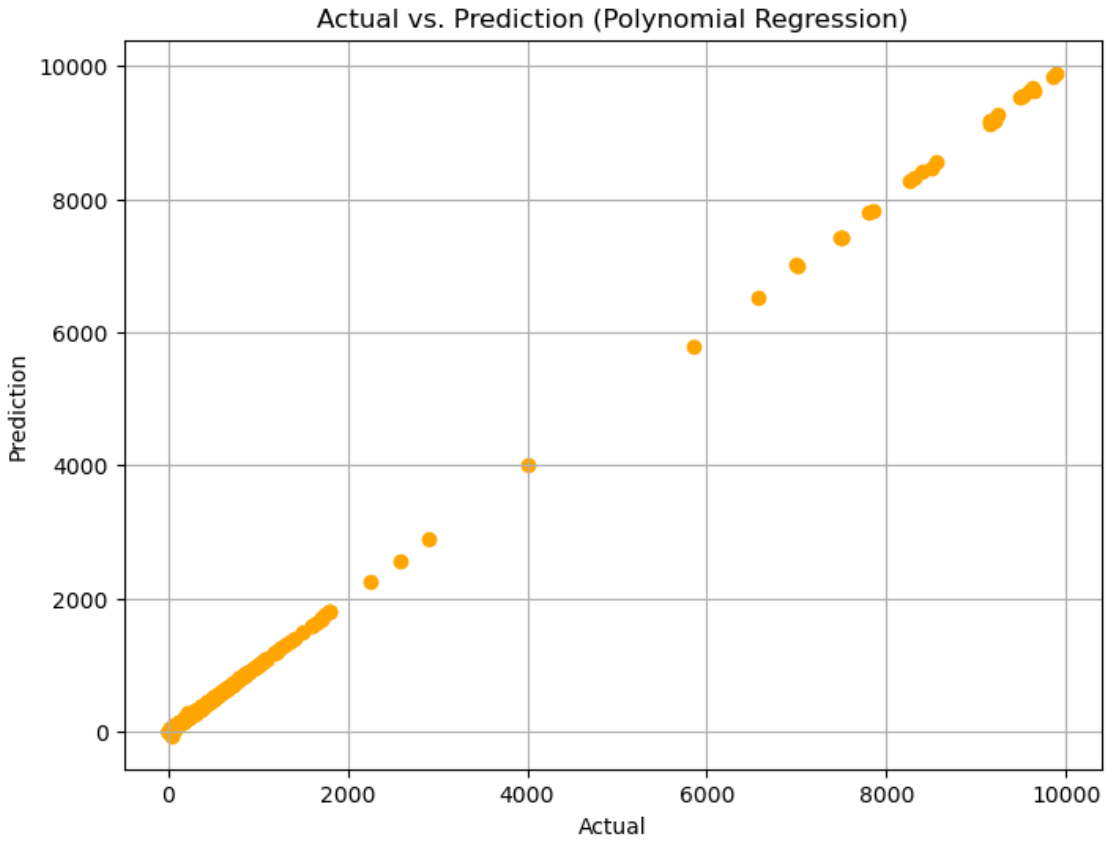
```
print(f"Decision Tree Regression R-squared: {dtR2}")

# Plot predictions vs. actual values for Decision Tree Regression
predictions_dt = dtPredictions.select("prediction").rdd.map(lambda row: row[0]).
  ↪collect()
actual_values_dt = dtPredictions.select(num_cols[8]).rdd.map(lambda row:␣
  ↪row[0]).collect()

mpt.figure(figsize=(8, 6))
mpt.scatter(actual_values_dt, predictions_dt, color='purple')
mpt.title("Actual vs. Prediction (Decision Tree Regression)")
mpt.xlabel("Actual")
mpt.ylabel("Prediction")
mpt.grid(True)
mpt.show()
```

```
Decision Tree Regression RMSE: 126.40519404146539
Decision Tree Regression MAE: 14.757403135970984
Decision Tree Regression R-squared: 0.9548177125166724
```



[18]:
```
# Polynomial Regression
poly_degree = 2  # Adjust the polynomial degree as needed
poly_expansion = PolynomialExpansion(degree=poly_degree, inputCol="features",␣
 ↪outputCol="poly_features")
record_poly = poly_expansion.transform(record)

# Split the polynomial dataset into training and testing sets
Train_x_poly, Test_y_poly = record_poly.randomSplit([0.8, 0.2], seed=42)

# Perform polynomial regression analysis
polyLr = LinearRegression(featuresCol="poly_features", labelCol=num_cols[8])
polyLrModel = polyLr.fit(Train_x_poly)
polyLrPredictions = polyLrModel.transform(Test_y_poly)

# Evaluate the polynomial regression model
polyLrRMSE, polyLrMAE, polyLrR2 = evaluate_model(polyLrPredictions, num_cols[8])
print(f"Polynomial Regression RMSE: {polyLrRMSE}")
print(f"Polynomial Regression MAE: {polyLrMAE}")
print(f"Polynomial Regression R-squared: {polyLrR2}")

# Plot predictions vs. actual values for Polynomial Regression
predictions_poly = polyLrPredictions.select("prediction").rdd.map(lambda row:␣
 ↪row[0]).collect()
actual_values_poly = polyLrPredictions.select(num_cols[8]).rdd.map(lambda row:␣
 ↪row[0]).collect()

mpt.figure(figsize=(8, 6))
mpt.scatter(actual_values_poly, predictions_poly, color='orange')
mpt.title("Actual vs. Prediction (Polynomial Regression)")
mpt.xlabel("Actual")
mpt.ylabel("Prediction")
mpt.grid(True)
mpt.show()
```

```
24/04/03 04:23:33 WARN Instrumentation: [07dcd2e5] regParam is zero, which might
cause numerical instability and overfitting.
24/04/03 04:23:40 WARN Instrumentation: [07dcd2e5] Cholesky solver failed due to
singular covariance matrix. Retrying with Quasi-Newton solver.

Polynomial Regression RMSE: 3.8514443191895453
Polynomial Regression MAE: 1.528240354659093
Polynomial Regression R-squared: 0.9999580544761655
```



[19]:
```
# Cross-Validation and Hyperparameter Tuning
# Define parameter grid for Random Forest
param_grid = ParamGridBuilder() \
    .addGrid(RF.maxDepth, [5, 10, 15]) \
    .addGrid(RF.numTrees, [10, 20, 30]) \
    .build()

# Define cross-validator
crossV = CrossValidator(estimator=RF, estimatorParamMaps=param_grid,␣
 ↪evaluator=RE(labelCol=num_cols[8], predictionCol="prediction",␣
 ↪metricName="rmse"), numFolds=3)

# Run cross-validation
cvModel = crossV.fit(Train_x)

# Get best model from cross-validation
bestRFModel = cvModel.bestModel

# Make predictions using best model
bestRF_predictions = bestRFModel.transform(Test_y)
```

```
# Evaluate best model
bestRFRMSE, bestRFMAE, bestRFR2 = evaluate_model(bestRF_predictions,␣
 ↪num_cols[8])
print(f"Best Random Forest Regression RMSE: {bestRFRMSE}")
print(f"Best Random Forest Regression MAE: {bestRFMAE}")
print(f"Best Random Forest Regression R-squared: {bestRFR2}")

# Plot predictions vs. actual values for Best Random Forest Regression
predictions_bestRF = bestRF_predictions.select("prediction").rdd.map(lambda row:
 ↪ row[0]).collect()
actual_values_bestRF = bestRF_predictions.select(num_cols[8]).rdd.map(lambda␣
 ↪row: row[0]).collect()

mpt.figure(figsize=(8, 6))
mpt.scatter(actual_values_bestRF, predictions_bestRF, color='red')
mpt.title("Actual vs. Prediction (Best Random Forest Regression)")
mpt.xlabel("Actual")
mpt.ylabel("Prediction")
mpt.grid(True)
mpt.show()
```

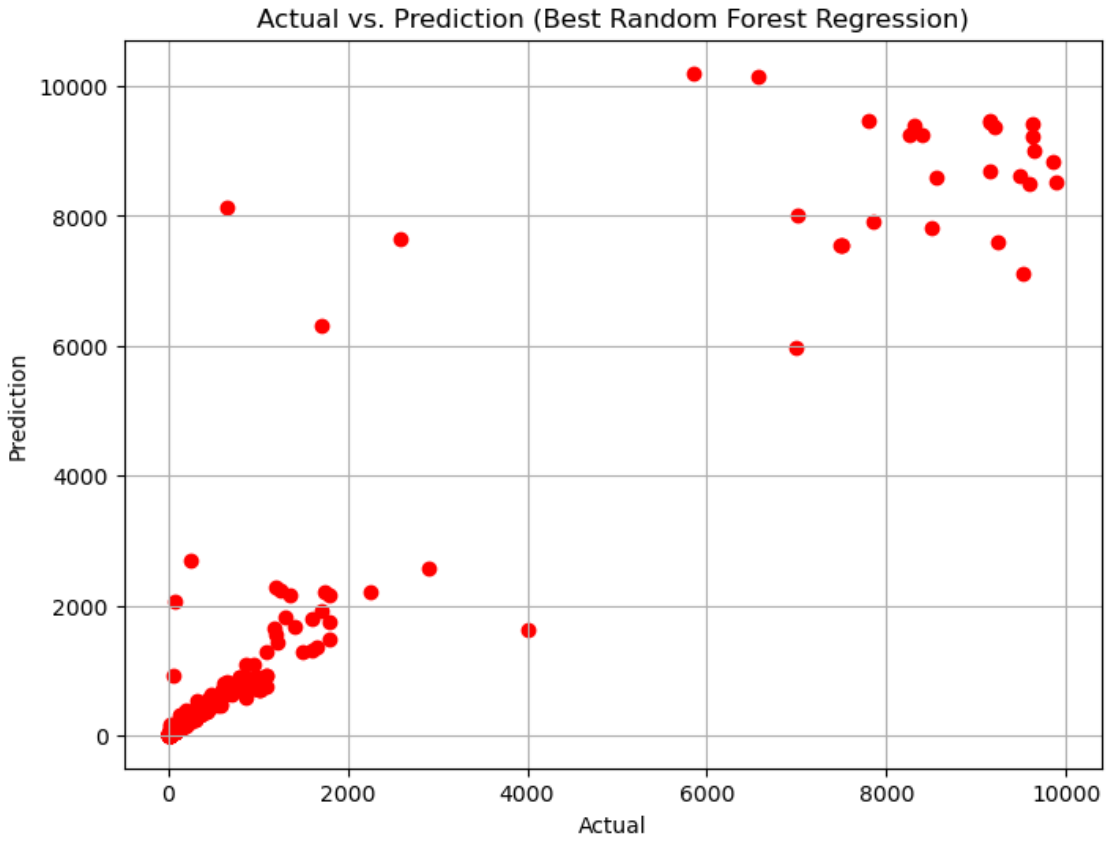
```
24/04/03 04:24:09 WARN DAGScheduler: Broadcasting large task binary with size
1044.2 KiB
24/04/03 04:24:09 WARN DAGScheduler: Broadcasting large task binary with size
1611.7 KiB
24/04/03 04:24:13 WARN DAGScheduler: Broadcasting large task binary with size
1475.9 KiB
24/04/03 04:24:13 WARN DAGScheduler: Broadcasting large task binary with size
2.2 MiB
24/04/03 04:24:16 WARN DAGScheduler: Broadcasting large task binary with size
1281.0 KiB
24/04/03 04:24:17 WARN DAGScheduler: Broadcasting large task binary with size
1896.9 KiB
24/04/03 04:24:18 WARN DAGScheduler: Broadcasting large task binary with size
2.7 MiB
24/04/03 04:24:18 WARN DAGScheduler: Broadcasting large task binary with size
3.8 MiB
24/04/03 04:24:20 WARN DAGScheduler: Broadcasting large task binary with size
5.1 MiB
24/04/03 04:24:23 WARN DAGScheduler: Broadcasting large task binary with size
1044.2 KiB
24/04/03 04:24:23 WARN DAGScheduler: Broadcasting large task binary with size
1611.7 KiB
24/04/03 04:24:24 WARN DAGScheduler: Broadcasting large task binary with size
2.4 MiB
24/04/03 04:24:25 WARN DAGScheduler: Broadcasting large task binary with size
```

```
3.6 MiB
24/04/03 04:24:27 WARN DAGScheduler: Broadcasting large task binary with size
5.3 MiB
24/04/03 04:24:28 WARN DAGScheduler: Broadcasting large task binary with size
7.5 MiB
24/04/03 04:24:29 WARN DAGScheduler: Broadcasting large task binary with size
1114.7 KiB
24/04/03 04:24:31 WARN DAGScheduler: Broadcasting large task binary with size
10.2 MiB
24/04/03 04:24:32 WARN DAGScheduler: Broadcasting large task binary with size
1338.9 KiB
24/04/03 04:24:37 WARN DAGScheduler: Broadcasting large task binary with size
1475.9 KiB
24/04/03 04:24:37 WARN DAGScheduler: Broadcasting large task binary with size
2.2 MiB
24/04/03 04:24:38 WARN DAGScheduler: Broadcasting large task binary with size
3.4 MiB
24/04/03 04:24:40 WARN DAGScheduler: Broadcasting large task binary with size
5.2 MiB
24/04/03 04:24:42 WARN DAGScheduler: Broadcasting large task binary with size
7.6 MiB
24/04/03 04:24:43 WARN DAGScheduler: Broadcasting large task binary with size
1229.4 KiB
24/04/03 04:24:45 WARN DAGScheduler: Broadcasting large task binary with size
10.7 MiB
24/04/03 04:24:47 WARN DAGScheduler: Broadcasting large task binary with size
1578.9 KiB
24/04/03 04:24:49 WARN DAGScheduler: Broadcasting large task binary with size
14.6 MiB
24/04/03 04:24:50 WARN DAGScheduler: Broadcasting large task binary with size
1916.3 KiB
24/04/03 04:25:01 WARN DAGScheduler: Broadcasting large task binary with size
1028.4 KiB
24/04/03 04:25:01 WARN DAGScheduler: Broadcasting large task binary with size
1571.7 KiB
24/04/03 04:25:05 WARN DAGScheduler: Broadcasting large task binary with size
1432.4 KiB
24/04/03 04:25:06 WARN DAGScheduler: Broadcasting large task binary with size
2.2 MiB
24/04/03 04:25:09 WARN DAGScheduler: Broadcasting large task binary with size
1212.7 KiB
24/04/03 04:25:09 WARN DAGScheduler: Broadcasting large task binary with size
1789.0 KiB
24/04/03 04:25:10 WARN DAGScheduler: Broadcasting large task binary with size
2.5 MiB
24/04/03 04:25:11 WARN DAGScheduler: Broadcasting large task binary with size
3.5 MiB
24/04/03 04:25:12 WARN DAGScheduler: Broadcasting large task binary with size
```

```
4.8 MiB
24/04/03 04:25:15 WARN DAGScheduler: Broadcasting large task binary with size
1028.4 KiB
24/04/03 04:25:16 WARN DAGScheduler: Broadcasting large task binary with size
1571.7 KiB
24/04/03 04:25:16 WARN DAGScheduler: Broadcasting large task binary with size
2.3 MiB
24/04/03 04:25:17 WARN DAGScheduler: Broadcasting large task binary with size
3.5 MiB
24/04/03 04:25:19 WARN DAGScheduler: Broadcasting large task binary with size
5.1 MiB
24/04/03 04:25:21 WARN DAGScheduler: Broadcasting large task binary with size
7.2 MiB
24/04/03 04:25:21 WARN DAGScheduler: Broadcasting large task binary with size
1090.0 KiB
24/04/03 04:25:23 WARN DAGScheduler: Broadcasting large task binary with size
9.9 MiB
24/04/03 04:25:24 WARN DAGScheduler: Broadcasting large task binary with size
1308.6 KiB
24/04/03 04:25:28 WARN DAGScheduler: Broadcasting large task binary with size
1432.4 KiB
24/04/03 04:25:29 WARN DAGScheduler: Broadcasting large task binary with size
2.2 MiB
24/04/03 04:25:30 WARN DAGScheduler: Broadcasting large task binary with size
3.3 MiB
24/04/03 04:25:31 WARN DAGScheduler: Broadcasting large task binary with size
5.0 MiB
24/04/03 04:25:33 WARN DAGScheduler: Broadcasting large task binary with size
7.3 MiB
24/04/03 04:25:34 WARN DAGScheduler: Broadcasting large task binary with size
1190.0 KiB
24/04/03 04:25:36 WARN DAGScheduler: Broadcasting large task binary with size
10.3 MiB
24/04/03 04:25:37 WARN DAGScheduler: Broadcasting large task binary with size
1544.2 KiB
24/04/03 04:25:40 WARN DAGScheduler: Broadcasting large task binary with size
14.2 MiB
24/04/03 04:25:42 WARN DAGScheduler: Broadcasting large task binary with size
1877.5 KiB
24/04/03 04:25:55 WARN DAGScheduler: Broadcasting large task binary with size
1001.6 KiB
24/04/03 04:25:56 WARN DAGScheduler: Broadcasting large task binary with size
1539.8 KiB
24/04/03 04:25:59 WARN DAGScheduler: Broadcasting large task binary with size
1456.8 KiB
24/04/03 04:26:00 WARN DAGScheduler: Broadcasting large task binary with size
2.2 MiB
24/04/03 04:26:03 WARN DAGScheduler: Broadcasting large task binary with size
```

```
1261.1 KiB
24/04/03 04:26:03 WARN DAGScheduler: Broadcasting large task binary with size
1871.7 KiB
24/04/03 04:26:04 WARN DAGScheduler: Broadcasting large task binary with size
2.7 MiB
24/04/03 04:26:05 WARN DAGScheduler: Broadcasting large task binary with size
3.8 MiB
24/04/03 04:26:06 WARN DAGScheduler: Broadcasting large task binary with size
5.2 MiB
24/04/03 04:26:09 WARN DAGScheduler: Broadcasting large task binary with size
1001.6 KiB
24/04/03 04:26:10 WARN DAGScheduler: Broadcasting large task binary with size
1539.8 KiB
24/04/03 04:26:11 WARN DAGScheduler: Broadcasting large task binary with size
2.3 MiB
24/04/03 04:26:11 WARN DAGScheduler: Broadcasting large task binary with size
3.5 MiB
24/04/03 04:26:13 WARN DAGScheduler: Broadcasting large task binary with size
5.1 MiB
24/04/03 04:26:15 WARN DAGScheduler: Broadcasting large task binary with size
7.2 MiB
24/04/03 04:26:16 WARN DAGScheduler: Broadcasting large task binary with size
1108.5 KiB
24/04/03 04:26:17 WARN DAGScheduler: Broadcasting large task binary with size
10.0 MiB
24/04/03 04:26:18 WARN DAGScheduler: Broadcasting large task binary with size
1335.6 KiB
24/04/03 04:26:23 WARN DAGScheduler: Broadcasting large task binary with size
1456.8 KiB
24/04/03 04:26:24 WARN DAGScheduler: Broadcasting large task binary with size
2.2 MiB
24/04/03 04:26:25 WARN DAGScheduler: Broadcasting large task binary with size
3.3 MiB
24/04/03 04:26:26 WARN DAGScheduler: Broadcasting large task binary with size
5.0 MiB
24/04/03 04:26:28 WARN DAGScheduler: Broadcasting large task binary with size
7.2 MiB
24/04/03 04:26:29 WARN DAGScheduler: Broadcasting large task binary with size
1166.5 KiB
24/04/03 04:26:31 WARN DAGScheduler: Broadcasting large task binary with size
10.2 MiB
24/04/03 04:26:32 WARN DAGScheduler: Broadcasting large task binary with size
1516.6 KiB
24/04/03 04:26:38 WARN DAGScheduler: Broadcasting large task binary with size
14.0 MiB
24/04/03 04:26:40 WARN DAGScheduler: Broadcasting large task binary with size
1881.9 KiB
24/04/03 04:26:48 WARN DAGScheduler: Broadcasting large task binary with size
```

```
1338.0 KiB
24/04/03 04:26:49 WARN DAGScheduler: Broadcasting large task binary with size
1987.5 KiB
24/04/03 04:26:50 WARN DAGScheduler: Broadcasting large task binary with size
2.8 MiB
24/04/03 04:26:51 WARN DAGScheduler: Broadcasting large task binary with size
4.0 MiB
24/04/03 04:26:52 WARN DAGScheduler: Broadcasting large task binary with size
5.5 MiB

Best Random Forest Regression RMSE: 175.17778205397764
Best Random Forest Regression MAE: 14.447615142913673
Best Random Forest Regression R-squared: 0.9132246404110798
```



[ ]: