Predicting the Past: Photometric Redshift Estimation on SDSS Data

Name: Akshay Atam

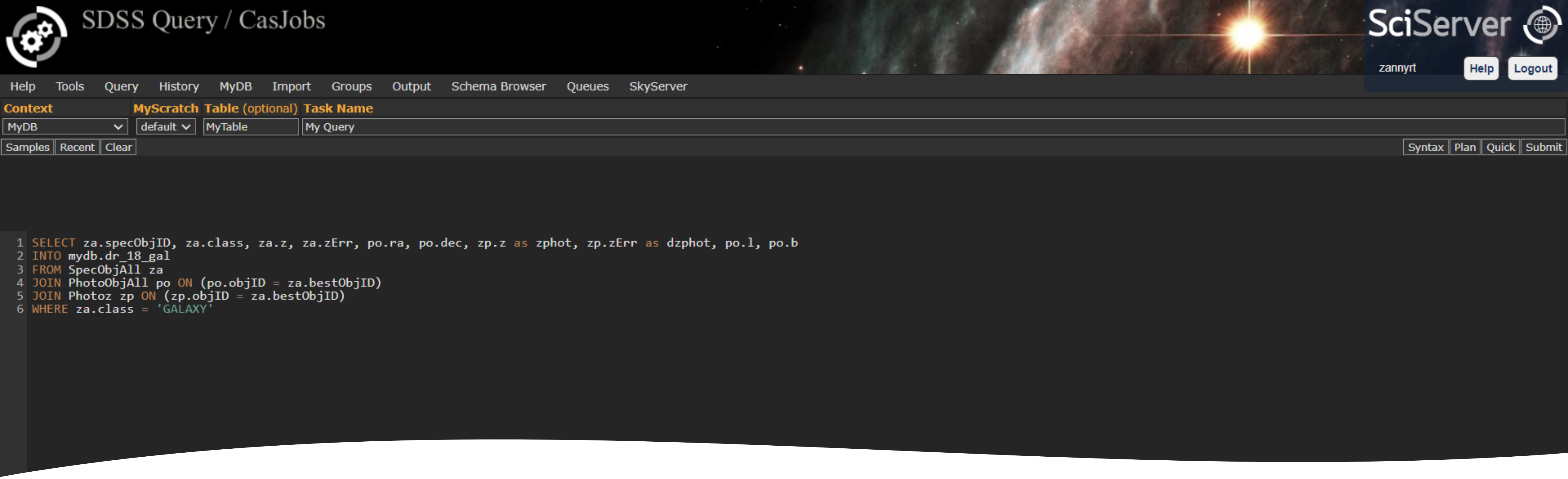
CWID: 20016304

Project Github link: https://github.com/akshay-atam/photometric-redshift-estimation/blob/main/scale_up_scale_out.ipynb

Hubble Ultra Deep Field

Problem Statement and Objective

- Problem Statement: Photometric Redshift estimation using SDSS ‘Galaxy’ data
- Objective: Develop a high-performing computing solution for handling the growing volume of astronomical data

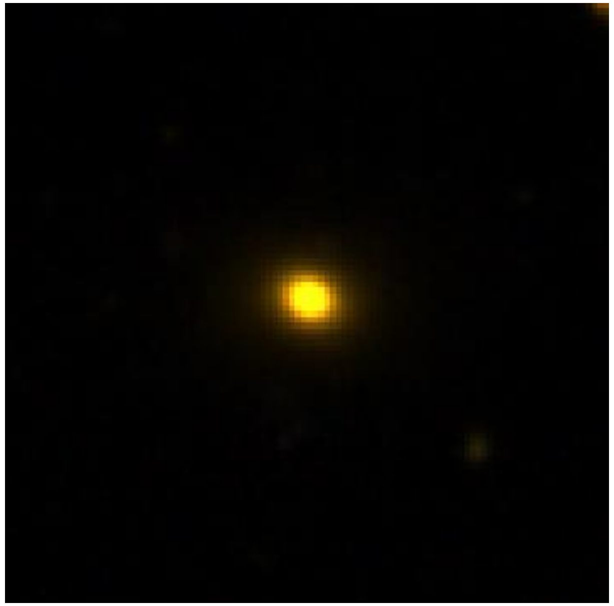


Data

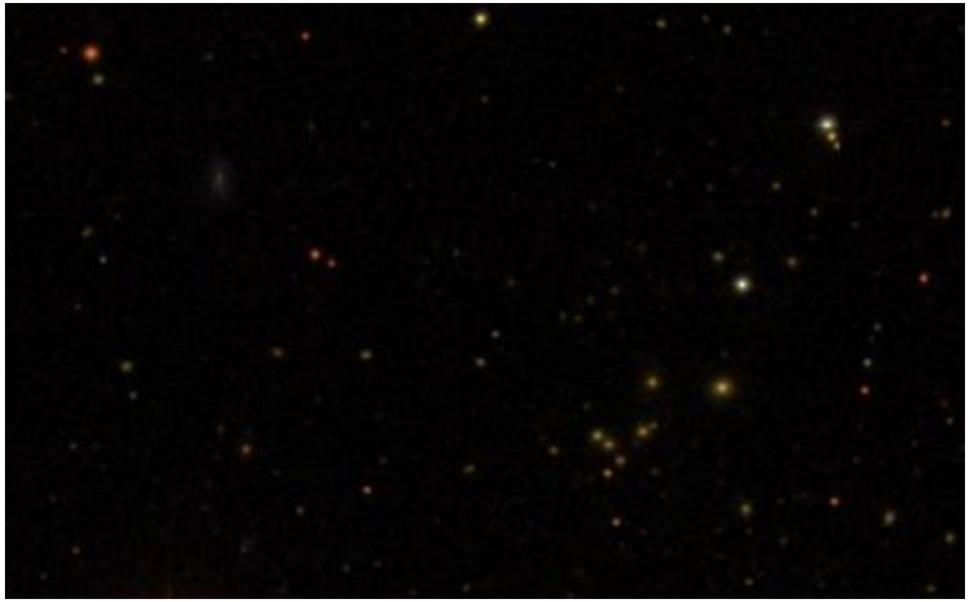
- Data Source: SDSS ‘Galaxy’ data from Data Release 18 (January 2023)
  - Unrestricted analysis with no limit on spectroscopic redshift value (reference paper used spectroscopic value <= 0.4 and Data Release 12)
  - Nearly six-fold increase in data size
- 2,924,493 examples (~3 million records)

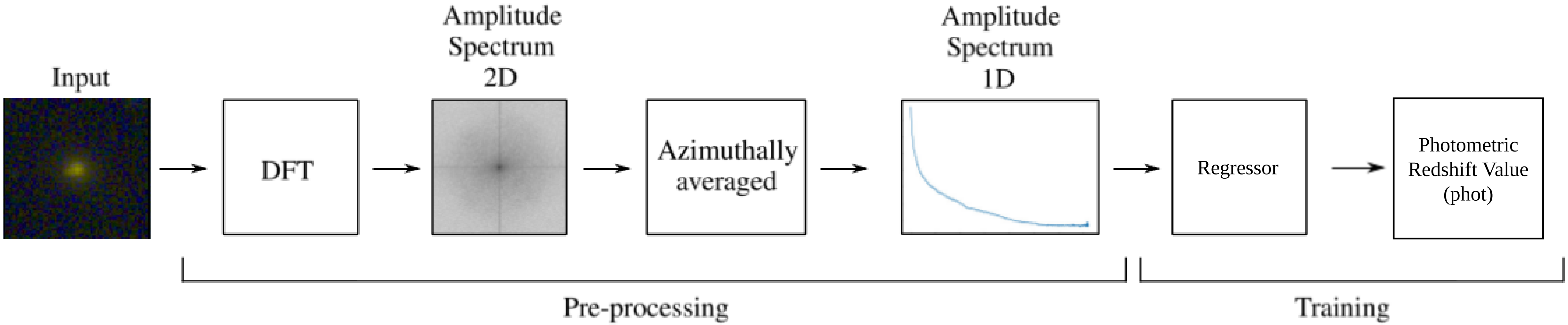
Problems/Issues faced

- The original proposal intended to use imaging data for using a Discrete Fourier Transform to estimate photometric redshift.
- Images not centered and calibrated.



vs.





Solution? Petrosian Magnitude

- Petrosian magnitude for each filter (ugriz)
  - Measurement of galaxy fluxes within a circular aperture whose radius is defined by the shape of the azimuthally averaged light profile.
  - Basically a 1-D spectrum of galaxy (like one I would've used with clean images!)
- Drawback
  - Petrosian magnitude creates the images in question (that was needed) - needs to be checked, verified and calibrated with other factors
  - Because the restriction on redshift value was lifted (to incorporate more examples) using Petrosian magnitude is the next best option.

SDSS Query / CasJobs

SciServer

zannyrt Help Logout

Help Tools Query History MyDB Import Groups Output Schema Browser Queues SkyServer

Context: MyDB | MyScratch: default | Table (optional): MyTable | Task Name: My Query

Samples Recent Clear | Syntax Plan Quick Submit

```sql
SELECT za.specObjID, za.class, za.z, za.zErr, po.ra, po.dec, zp.z as zphot, zp.zErr as dzphot, po.l, po.b
INTO mydb.dr_18_gal
FROM SpecObjAll za
JOIN PhotoObjAll po ON (po.objID = za.bestObjID)
JOIN Photoz zp ON (zp.objID = za.bestObjID)
WHERE za.class = 'GALAXY'
```

- specObjID: ID (used as key)
- class: Respective class (‘GALAXY’, ‘QSO’, or ‘STAR’); here class is ‘GALAXY’
- z and zErr: Spectroscopic Redshift and error in Spectroscopic Redshift
- ra: Right ascension of galaxy
- dec: Declination of galaxy
- zphot and dzphot: Photometric Redshift and error in Photometric Redshift

Context: MyDB | MyScratch: default | Table (optional): MyTable | Task Name: My Query

Samples | Recent | Clear | Syntax | Plan | Quick | Submit

```sql
SELECT za.specObjID, po.petroMag_u, po.petroMag_g, po.petroMag_r, po.petroMag_i, po.petroMag_z, po.petroMagErr_u, po.petroMagErr_g, po.petroMagErr_r, po.petroMagErr_i, po.petroMagErr_z
INTO mydb.dr_18_gal_petro
FROM SpecObjAll za
JOIN PhotoObjAll po ON (po.objID = za.bestObjID)
JOIN Photoz zp ON (zp.objID = za.bestObjID)
WHERE za.class = 'GALAXY'
```

- specObjID: ID (used as key)
- petroMag: Petrosian magnitude in different wavelengths
- petroMagErr: Error in Petrosian magnitude in different wavelengths
- l: Galactic Latitude
- b: Galactic Longitude

Dataset Schema

- All necessary columns not possible to extract in one go through CasJobs
- Divided into two files:
  - gal_info: Necessary information for the galaxies
  - gal_petro: Petrosian information in different wavelengths.

```
gal_info_schema = T.StructType([
    T.StructField('specObjID', T.LongType(), True),
    T.StructField('class', T.StringType(), True),
    T.StructField('z', T.FloatType(), True),
    T.StructField('zErr', T.FloatType(), True),
    T.StructField('ra', T.FloatType(), True),
    T.StructField('dec', T.FloatType(), True),
    T.StructField('zphot', T.FloatType(), True),
    T.StructField('dzphot', T.FloatType(), True),
    T.StructField('l', T.FloatType(), True),
    T.StructField('b', T.FloatType(), True),
])

gal_petro_schema = T.StructType([
    T.StructField('specObjID', T.LongType(), True),
    T.StructField('petroMag_u', T.FloatType(), True),
    T.StructField('petroMag_g', T.FloatType(), True),
    T.StructField('petroMag_r', T.FloatType(), True),
    T.StructField('petroMag_i', T.FloatType(), True),
    T.StructField('petroMag_z', T.FloatType(), True),
    T.StructField('petroMagErr_u', T.FloatType(), True),
    T.StructField('petroMagErr_g', T.FloatType(), True),
    T.StructField('petroMagErr_r', T.FloatType(), True),
    T.StructField('petroMagErr_i', T.FloatType(), True),
    T.StructField('petroMagErr_z', T.FloatType(), True),
])
```

Experimentation

- Need two achieve three objectives
  - Select best hyperparameters
  - Track performance with varying dataset size
  - Scaling out strategies
- Models used
  - Linear Regression
  - Decision Tree Regression
  - Random Forest Regressor
  - Gradient Boosted Tree Regressor

Solution for Experimentation

- Perform 5-fold Cross Validation using different combinations of model parameters for best hyperparameters
- Train the best hyperparameters on varying model sizes
    - 25%
    - 50%
    - 75%
    - 100%

```
grid_search = (
    ParamGridBuilder()
    .addGrid(lr.regParam, [0.0, 0.2])
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])
    .addGrid(lr.loss, ['squaredError'])
    .build()
)

cv = CrossValidator(
    estimator=pipeline,
    estimatorParamMaps=grid_search,
    evaluator=evaluator,
    numFolds=5,
    seed=42,
    collectSubModels=True
)

cv_model = cv.fit(train_data)
```

Dataset Sampling and Evaluation Metrics

- Sampling done using ‘pyspark.sql.DataFrame.sample’
  - We can define if sampling is to be done with replacement or not
  - Defining the fraction of DataFrame to sample
  - Also set seed
- Evaluation Metrics
  - Root Mean Squared Error (RMSE)
  - R^2 score (R2)

```python
evaluation_results = []
percentage_list = [0.25, 0.5, 0.75, 1.0]

# Loop through different percentages
for percentage in percentage_list:
    start = time.time()
    sampled_data = gal_data.sample(fraction=percentage, seed=42)

    train_data, test_data = sampled_data.randomSplit([0.8, 0.2], seed=42)
    model = pipeline.fit(train_data)
    predictions = model.transform(test_data)

    # Evaluation
    evaluator1 = RegressionEvaluator(labelCol="zphot", predictionCol="prediction", metricName="r2")
    r2 = evaluator1.evaluate(predictions)

    evaluator2 = RegressionEvaluator(labelCol="zphot", predictionCol="prediction", metricName="rmse")
    rmse = evaluator2.evaluate(predictions)

    # Append results to the list
    evaluation_results.append((percentage, r2, rmse))
    print(time.time() - start)

# Print the evaluation results
for percentage, r2, rmse in evaluation_results:
    print(f"Percentage of Data: {percentage * 100}% | R2: {r2} | RMSE: {rmse}")
```

Scaling out strategies

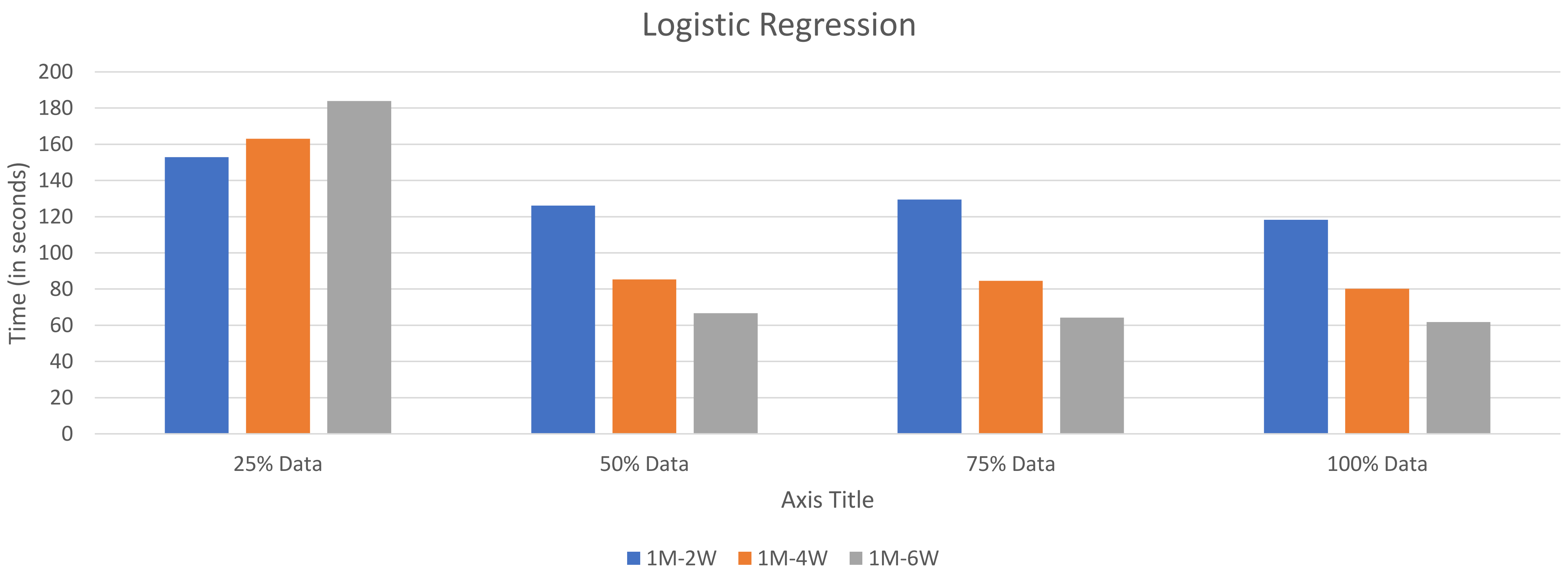
- Models trained on three configurations (with 4 different sizes of dataset)
- Bumping up worker nodes to 8 resulted in Insufficient 'IN_USE_ADDRESSES' quota

Master Node	Worker Nodes
1 x n2-standard-2	2 x n2-standard-2
1 x e2-standard-2	4 x e2-standard-2
1 x e2-standard-2	6 x e2-standard-2

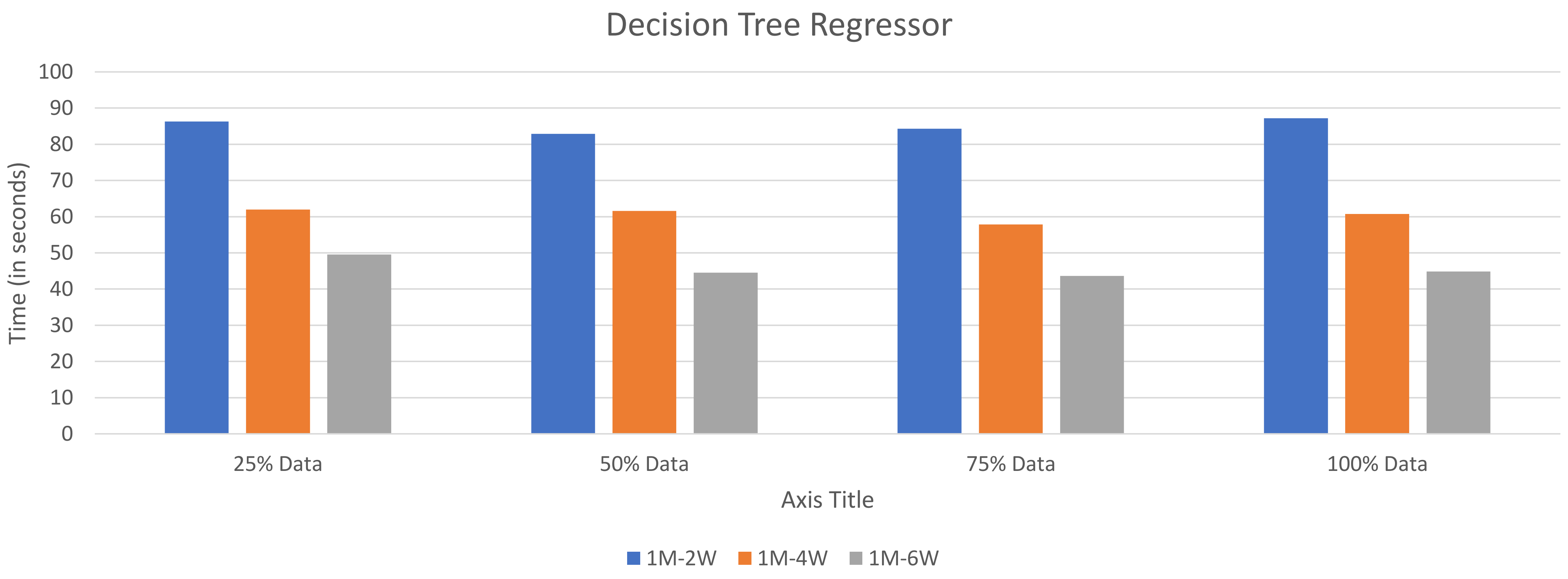
Results (Best Model Hyperparameters)

- Linear Regression
  - regParam = 0.2
  - elasticNetParam = 1.0
  - loss = ‘squaredError’
- Decision Tree Regressor
  - maxDepth = 5
- Random Forest Regressor
  - numTrees = 4
  - featureSubsetStrategy = ‘onethird’
- Gradient Boosted Tree Regressor
  - lossType = ‘squared’

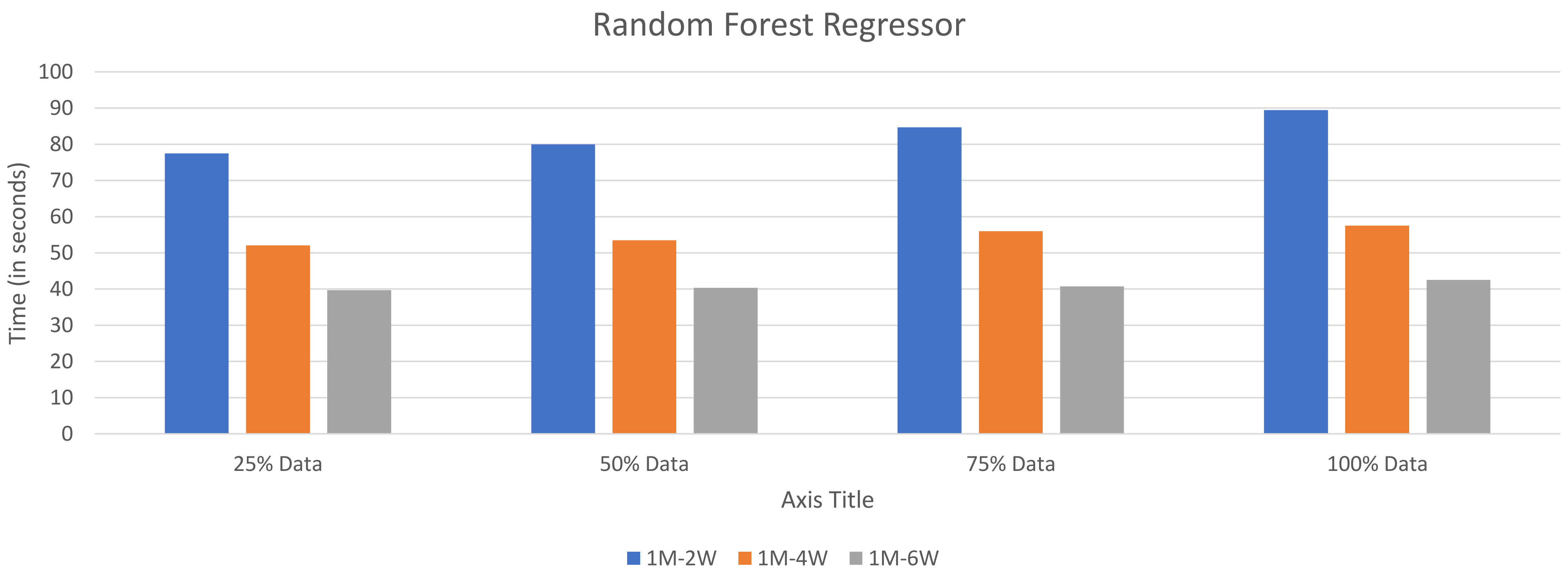
Result (Time of Execution)



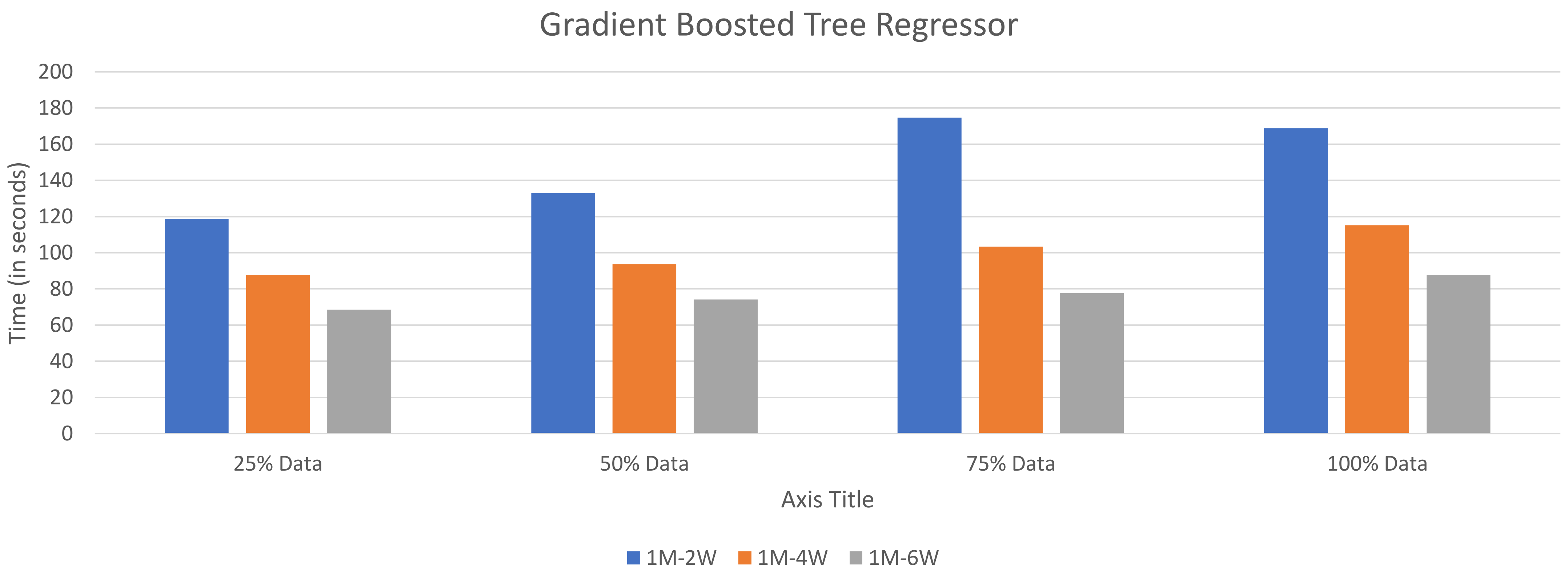
Result (Time of Execution)



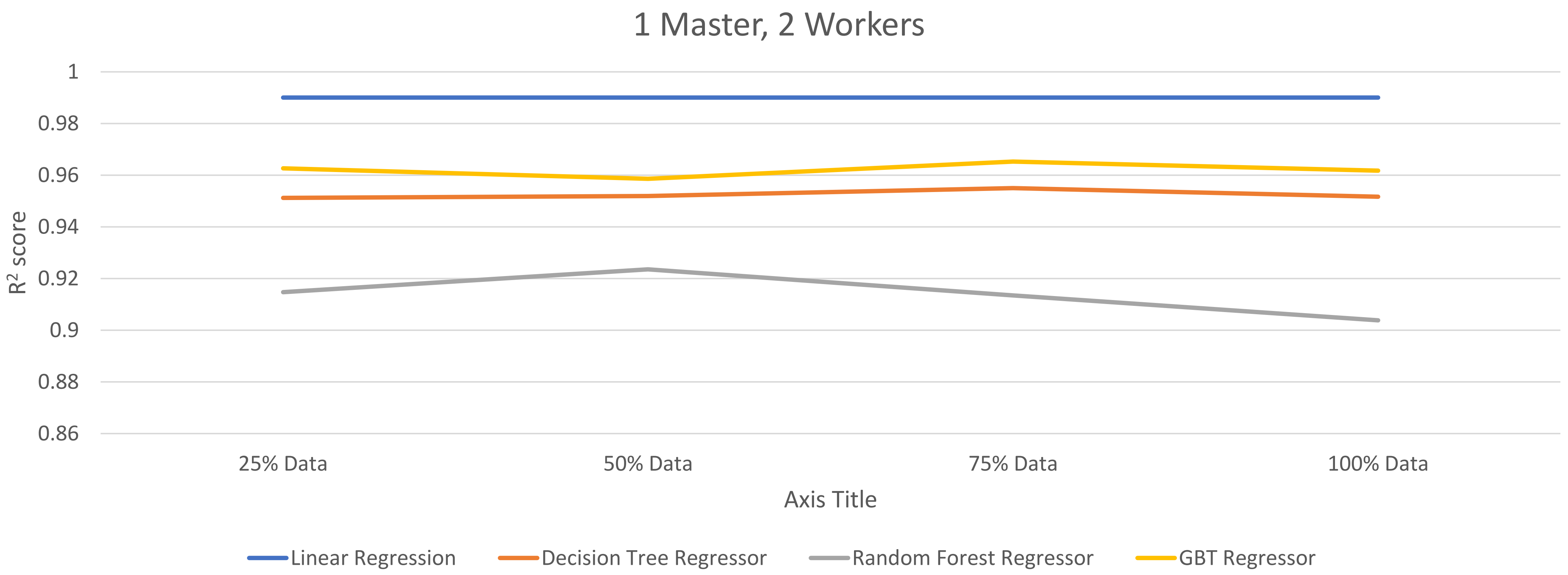
Result (Time of Execution)



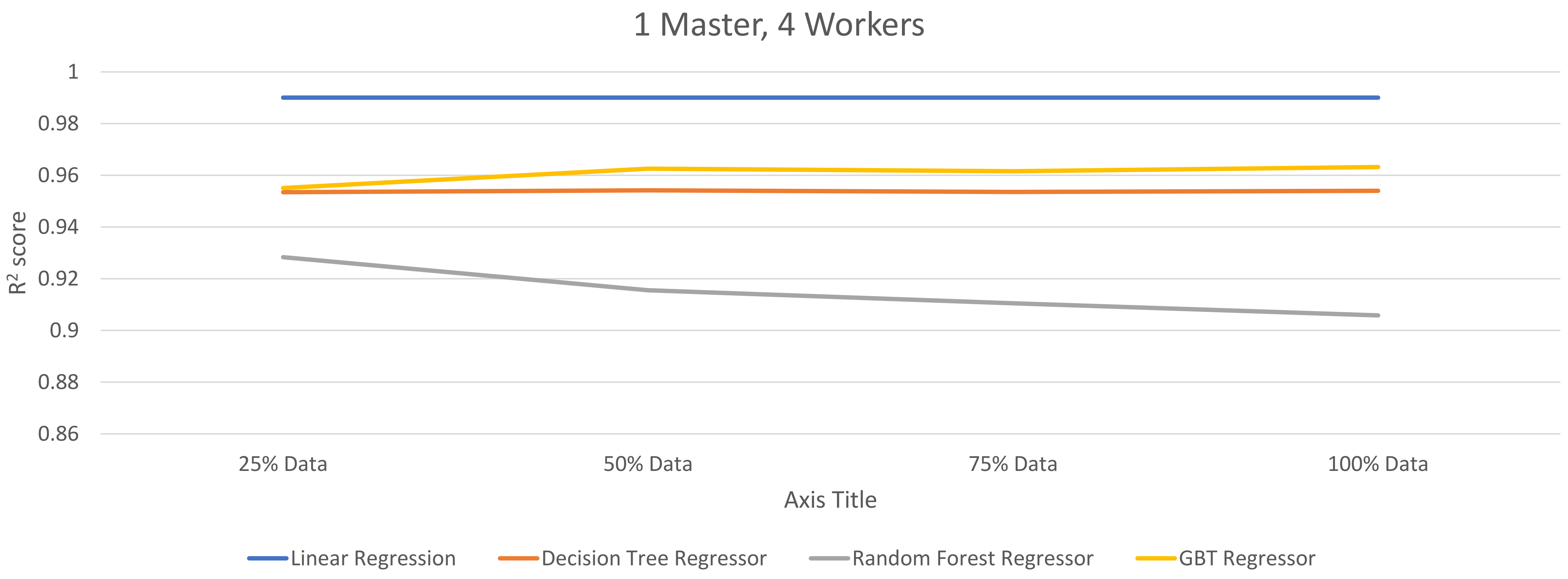
Result (Time of Execution)



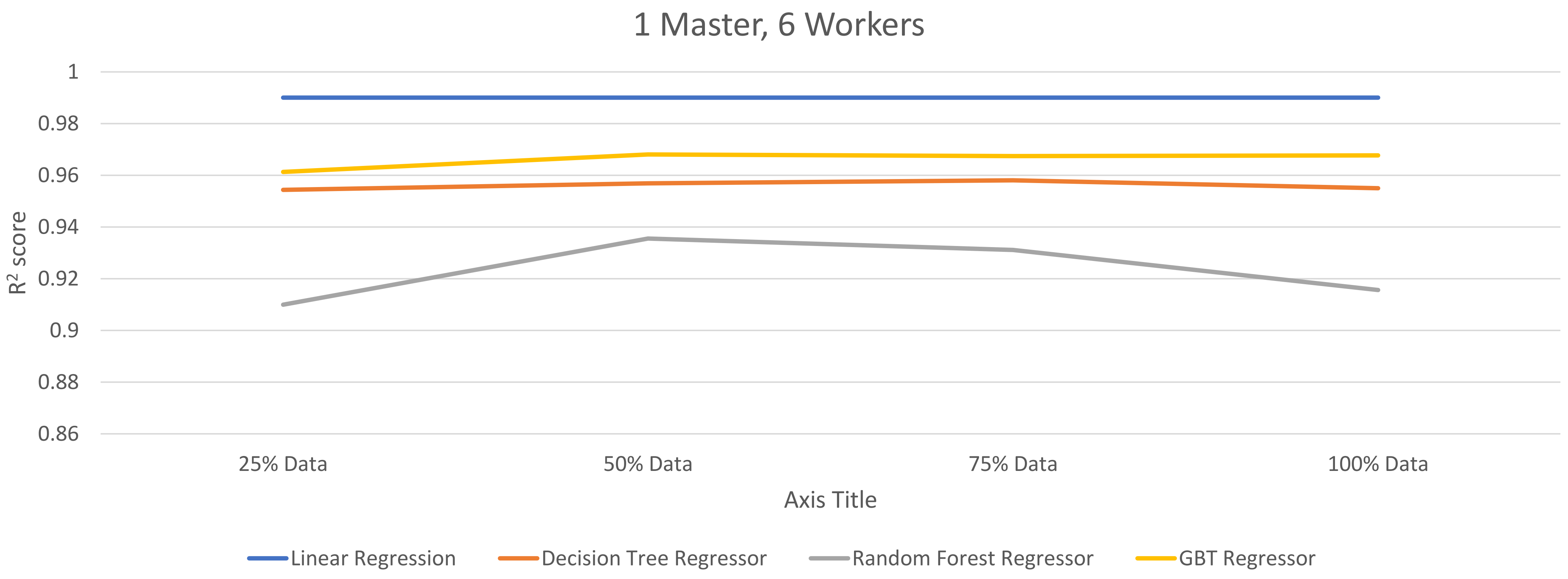
Result (R^2 score)



Result (R^2 score)



Result (R^2 score)



Result (RMSE, Linear Regression)

	1 Master, 2 Workers	1 Master, 4 Workers	1 Master, 6 Workers
25% Data	0.2815	0.281	0.281
50% Data	0.2843	0.284	0.283
75% Data	0.2809	0.2812	0.2846
100% Data	0.2796	0.2804	0.2837

Result (RMSE, Decision Tree Regression)

	1 Master, 2 Workers	1 Master, 4 Workers	1 Master, 6 Workers
25% Data	175.93	172.44	169.07
50% Data	177.70	173.82	167.05
75% Data	169.59	171.63	165.96
100% Data	173.46	169.92	171.36

Result (RMSE, Random Forest Regression)

	1 Master, 2 Workers	1 Master, 4 Workers	1 Master, 6 Workers
25% Data	232.48	214.15	238.50
50% Data	224.08	236.21	205.29
75% Data	235.19	238.42	212.83
100% Data	244.90	243.29	234.80

Result (RMSE, GBT Regression)

	1 Master, 2 Workers	1 Master, 4 Workers	1 Master, 6 Workers
25% Data	153.74	169.35	155.72
50% Data	164.81	157.10	143.71
75% Data	148.91	156.11	146.41
100% Data	154.16	151.93	145.17

Conclusion and Future Works

- Study of Galaxy data to estimate photometric redshift with spectroscopic and Petrosian data.
- Previous works show a near linear relationship between photometric redshift and spectroscopic redshift.
- Comparison of four regression models across three different cluster configuration with varying dataset size.
- Future work includes using image data to achieve the objective that was set.
- Analysis of linearity between spectroscopic and photometric results.

Questions?