New York Institute of Technology
College of Engineering and Computing Sciences

Regression Modelling for House Price Predictions

DTSC 701/CSCI 636 Big Data Analytics
Professor Liangwen Wu

Stevenson Chittumuri (1226890), Albert Chen (1299225)

Due Date: August 25th, 2023

Abstract—**This research project sits at the intersection of cloud computing, data architecture, and predictive modeling using California's housing data to predict housing prices. Our main objective is to run our regression models to ascertain trends in the data set that may provide insight into California's housing price distribution and second, to guide cloud service utilization and data storage optimization. We have provided a step-by-step guide to run the Spark application on AWS. Once up and running, three regression models were deployed in a production AWS environment: Linear Regression, Decision Tree Regression, and Random Forest Regression. Out of the set of regression models implemented, Random Forest Regression emerged as the most promising, with approximately 75.63% of the target variance being explained by the predictors and the least average prediction error. The study concludes by running the California Housing Data Set on a KNN Regression Model not available in our current production environment to offer more insight into the trends in the data set that require the integration of domain knowledge.**

Contents

I. Introduction

This project report aims to elucidate the complexities of cloud computing, data architecture decisions, and model development and evaluation, through data analysis and regression modeling of California's housing Data Set [2]. This report provides not just insights into the state's housing price distribution, but also offers a framework and guide for effectively leveraging cloud services and optimizing data storage choices. Our choice of the California Housing Data Set is intentional given its full-featured data set with low dimensionality allowing ease of usage in regression modeling deployed in a cloud production environment with data storage capabilities.

II. Data Storage Decision: Data Lake vs. Data Warehouse

When choosing a storage system, our decision came down to using a data lake or a data warehouse. AWS, our chosen cloud service, offers robust solutions for both. For data lakes, AWS provides Amazon S3 (Simple Storage Service), which is designed to store and retrieve any amount of data. S3 is reliable, scalable, and offers easy data access. Given its capabilities, it was an appealing choice. On the data warehouse side, AWS offers Amazon Redshift, a fully managed, petabyte-scale data warehouse service. Redshift is optimized for online analytic processing (OLAP) and business intelligence tools.

While Redshift offers fast querying capabilities, it's primarily structured for this type of workload.

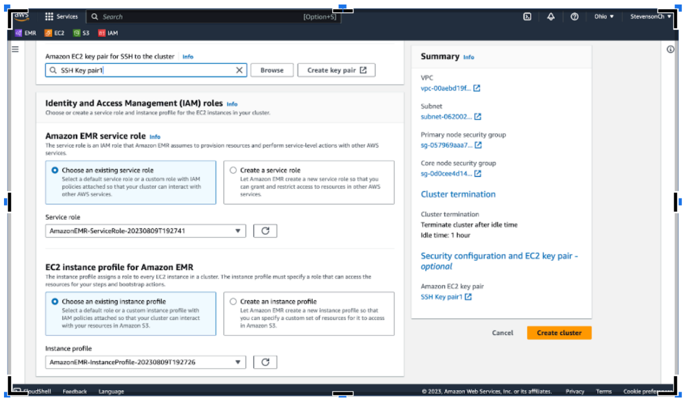
We specifically chose the data lake approach using Amazon S3 for our project. Due to its scalability, Amazon S3 easily scales to accommodate growing data without any upfront costs [3]. In terms of flexibility, S3 can store both structured and unstructured data. Given the diverse data types in our data set, this was a significant advantage. With regards to cost-effectiveness, Amazon S3 uses a pay-as-you-go model aligned well with our project's budget and goals. While Amazon Redshift is a powerful data warehouse option, we prioritized flexibility and scalability over purely fast querying capabilities for this particular project. Thus, the data lake approach using Amazon S3 was more in line with our needs.

III. Documentation for Running Spark Application on AWS

Initial Setup: Create an AWS account and have CSV and Python scripts downloaded.

Step 1: Setting Up Your EC2 Instance

- Log in to your AWS Management Console.
- Navigate to EMR on EC2 and 'Create Cluster'.
- Choose the 'Spark' application, and 'm5.xlarge' instance types for Primary, Core, and Tasks.
- Provision '3' for the instance sizes for Primary and Task.
- Find "Security configuration and EC2 key pair", and be sure to create or select an SSH Key Pair to remotely access the cluster from your machine. Be sure to have it downloaded onto your machine.
- Finally, create or select roles under the 'Identity and Access Management (IAM) roles' section.



Amazon EC2 key pair for SSH to the cluster
SSH Key pair1

Identity and Access Management (IAM) roles
Choose or create a service role and instance profile for the EC2 instances in your cluster.

Amazon EMR service role
The service role is an IAM role that Amazon EMR assumes to provision resources and perform service-level actions with other AWS services.

[✓ Choose an existing service role] Select a default service role or a custom role with IAM policies attached so that your cluster can interact with other AWS services.
Create a service role: Let Amazon EMR create a new service role so that you can grant and restrict access to resources in other AWS services.

Service role
AmazonEMR-ServiceRole-20230809T192741

EC2 instance profile for Amazon EMR
The instance profile assigns a role to every EC2 instance in a cluster. The instance profile must specify a role that can access the resources for your steps and bootstrap actions.

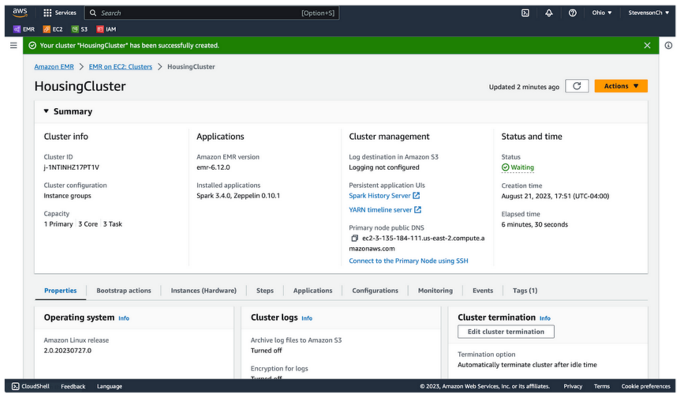
[✓ Choose an existing instance profile] Select a default role or a custom instance profile with IAM policies attached so that your cluster can interact with your resources in Amazon S3.
Create an instance profile: Let Amazon EMR create a new instance profile so that you can specify a custom set of resources for it to access in Amazon S3.

Instance profile
AmazonEMR-InstanceProfile-20230809T192726

Summary
VPC: vpc-00aebd19f...
Subnet: subnet-062002...
Primary node security group: sg-057969aaa7...
Core node security group: sg-0d0cee4d14...
Cluster termination
Cluster termination: Terminate cluster after idle time
Idle time: 1 hour
Security configuration and EC2 key pair - *optional*
Amazon EC2 key pair: SSH Key pair1

© 2023, Amazon Web Services, Inc. or its affiliates.

Fig. 1. Shows that there are no missing/misconfigured values on the right side, allowing me to create the cluster.



Your cluster "HousingCluster" has been successfully created.

Amazon EMR > EMR on EC2: Clusters > HousingCluster

HousingCluster
Updated 2 minutes ago

Summary

Cluster info	Applications	Cluster management	Status and time
Cluster ID: j-1NTINHZ17PT1V	Amazon EMR version: emr-6.12.0	Log destination in Amazon S3: Logging not configured	Status: Waiting
Cluster configuration: Instance groups	Installed applications: Spark 3.4.0, Zeppelin 0.10.1	Persistent application UIs: Spark History Server, YARN timeline server	Creation time: August 21, 2023, 17:51 (UTC-04:00)
Capacity: 1 Primary 3 Core 3 Task		Primary node public DNS: ec2-3-135-184-111.us-east-2.compute.amazonaws.com; Connect to the Primary Node using SSH	Elapsed time: 6 minutes, 30 seconds

Operating system
Amazon Linux release: 2.0.20230727.0

Cluster logs
Archive log files to Amazon S3: Turned off
Encryption for logs: Turned off

Cluster termination
Termination option: Automatically terminate cluster after idle time

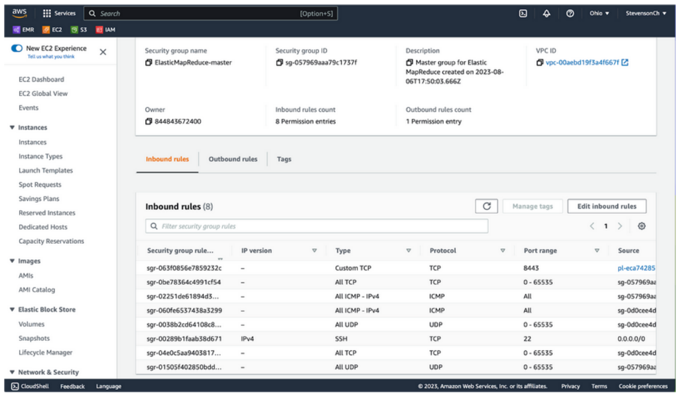
© 2023, Amazon Web Services, Inc. or its affiliates.

Fig. 2. Successfully created an online cluster with a public DNS node available for SSH

Step 2: Enable SSH privileges to EC2 Cluster

- When you open the cluster, scroll down to the "EC2 security groups (firewall)" section and open the hyperlink for your primary node's security group.
- Inside the "Inbound rules" section be sure to add SSH (port 22) privileges for your IP address.
- If you don't have SSH privileges, click on "Edit inbound rules", scroll down to "Add rule"

and choose SSH and the desired IPV4 address (preferably “0.0.0.0/0”) for global access as this is just a test. Save rules and you should see “SSH” as a rule in your primary group.



Security group name	Security group ID	Description	VPC ID
ElasticMapReduce-master	sg-057969aaa79c1737f	Master group for Elastic MapReduce created on 2023-08-06T17:50:03.666Z	vpc-00aebd19f3a4f667f

Owner	Inbound rules count	Outbound rules count
844843672400	8 Permission entries	1 Permission entry

Inbound rules (8)

Security group rule...	IP version	Type	Protocol	Port range	Source
sgr-063f0856e7859232c	–	Custom TCP	TCP	8443	pl-eca74285
sgr-0be78364c4991cf54	–	All TCP	TCP	0 - 65535	sg-057969aa
sgr-02251de61894d3...	–	All ICMP - IPv4	ICMP	All	sg-057969aa
sgr-060fe6537438a3299	–	All ICMP - IPv4	ICMP	All	sg-0d0cee4d
sgr-0038b2cd64108c8...	–	All UDP	UDP	0 - 65535	sg-0d0cee4d
sgr-00289b1faab38d671	IPv4	SSH	TCP	22	0.0.0.0/0
sgr-04e0c5aa9403817...	–	All TCP	TCP	0 - 65535	sg-0d0cee4d
sgr-01505f402850bdd...	–	All UDP	UDP	0 - 65535	sg-057969aa

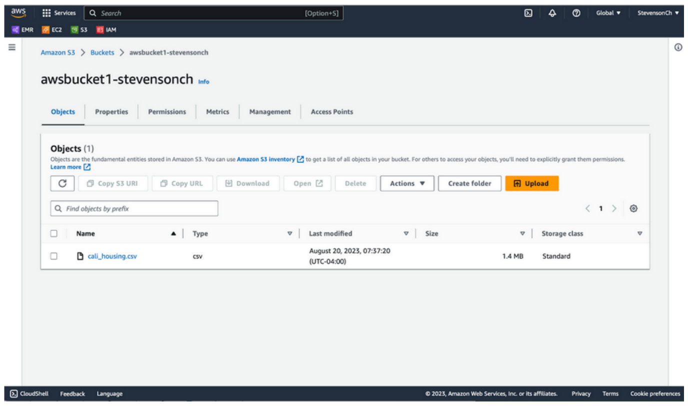
Fig. 3. SSH protocol is part of the inbound rules now

Step 3: Upload dataset to S3 Bucket (data lake)

- In AWS Console, open S3 and create a bucket, and uncheck “Block all public access” (if you don’t already have one dedicated to this project).

- Then inside this bucket, click on ’Upload’ and upload the CSV file containing the dataset.

- After uploading, select the file and click on ’Copy S3 URI’ and paste that link into the Python script in the read CSV method.



Amazon S3 > Buckets > awsbucket1-stevensonch

awsbucket1-stevensonch

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions.

Name	Type	Last modified	Size	Storage class
cali_housing.csv	csv	August 20, 2023, 07:37:20 (UTC-04:00)	1.4 MB	Standard

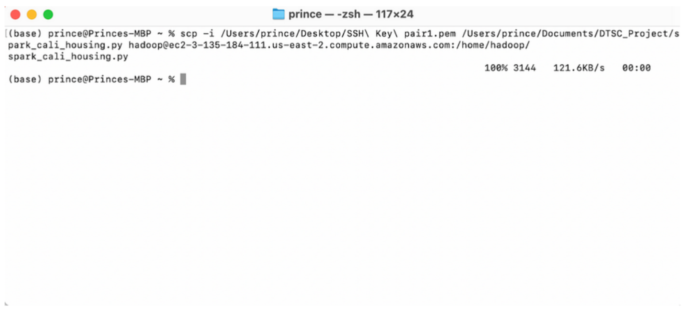
Fig. 4. File inside the data lake S3 bucket

Step 4: Upload Your Script to the Cluster

- Open the terminal (or command prompt) on your machine

- Use the ’scp’ command to securely copy your Python script to the EC2 instance:

scp -i /path/to/your/ssh_key.pem /path/to/spark_cali_housing.py hadoop@your-ec2-ip-address:/home/hadoop/



```
prince — -zsh — 117×24
(base) prince@Princes-MBP ~ % scp -i /Users/prince/Desktop/SSH\ Key\ pair1.pem /Users/prince/Documents/DTSC_Project/s
park_cali_housing.py hadoop@ec2-3-135-184-111.us-east-2.compute.amazonaws.com:/home/hadoop/
spark_cali_housing.py
                                                                                       100% 3144   121.6KB/s   00:00
(base) prince@Princes-MBP ~ % 
```

Result in terminal if ran successfully

Fig. 5. Result in terminal if ran successfully

Step 5: SSH into Your EC2 Instance

- Inside your terminal session, we will access your EC2 instance using the ‘ssh‘ command:

ssh -i /path/to/your/ssh_key.pem hadoop@your-ec2-ip-address

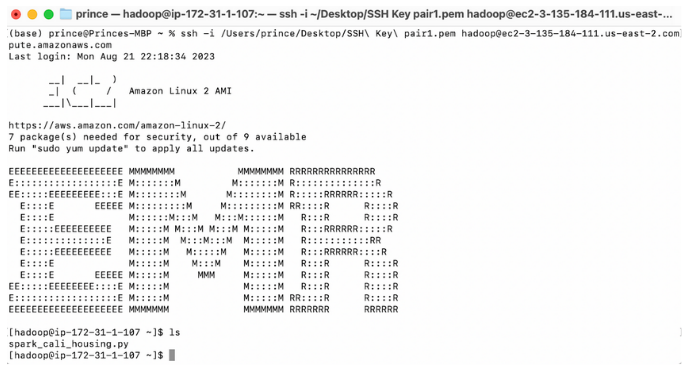
Fig. 6. SSH command

- After you are SSH’ed into the cluster, type ’ls’ to see if the Python script has been uploaded.

Step 6: Run Your Spark Application on the Cluster

- Execute your Spark application using the ‘spark-submit‘ command:

During execution, you should see Spark start processes and allocate resources to spark-related



prince — hadoop@ip-172-31-1-107:~ — ssh -i ~/Desktop/SSH Key pair1.pem hadoop@ec2-3-135-184-111.us-east-...

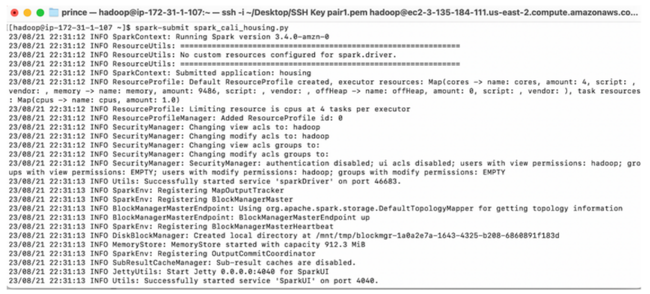
```
(base) prince@Princes-MBP ~ % ssh -i /Users/prince/Desktop/SSH\ Key\ pair1.pem hadoop@ec2-3-135-184-111.us-east-2.com
pute.amazonaws.com
Last login: Mon Aug 21 22:18:34 2023

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
7 package(s) needed for security, out of 9 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-1-107 ~]$ ls
spark_cali_housing.py
[hadoop@ip-172-31-1-107 ~]$ 
```

Fig. 7. Proof the SSH works and Python script has been uploaded

spark-submit spark_cali_housing.py

Fig. 8. Spark Submit Command

jobs. After completion, you'll see results for the models and their accuracy.



prince — hadoop@ip-172-31-1-107:~ — ssh -i ~/Desktop/SSH Key pair1.pem hadoop@ec2-3-135-184-111.us-east-2.compute.amazonaws.co...

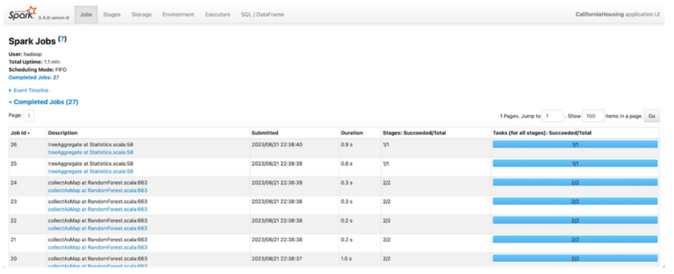
```
[hadoop@ip-172-31-1-107 ~]$ spark-submit spark_cali_housing.py
23/08/21 22:31:12 INFO SparkContext: Running Spark version 3.4.0-amzn-0
23/08/21 22:31:12 INFO ResourceUtils: ==============================================================
23/08/21 22:31:12 INFO ResourceUtils: No custom resources configured for spark.driver.
23/08/21 22:31:12 INFO ResourceUtils: ==============================================================
23/08/21 22:31:12 INFO SparkContext: Submitted application: housing
23/08/21 22:31:12 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount: 4, script: ,
vendor: , memory -> name: memory, amount: 9486, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources
: Map(cpus -> name: cpus, amount: 1.0)
23/08/21 22:31:12 INFO ResourceProfile: Limiting resource is cpus at 4 tasks per executor
23/08/21 22:31:12 INFO ResourceProfileManager: Added ResourceProfile id: 0
23/08/21 22:31:12 INFO SecurityManager: Changing view acls to: hadoop
23/08/21 22:31:12 INFO SecurityManager: Changing modify acls to: hadoop
23/08/21 22:31:12 INFO SecurityManager: Changing view acls groups to:
23/08/21 22:31:12 INFO SecurityManager: Changing modify acls groups to:
23/08/21 22:31:12 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: hadoop; gro
ups with view permissions: EMPTY; users with modify permissions: hadoop; groups with modify permissions: EMPTY
23/08/21 22:31:13 INFO Utils: Successfully started service 'sparkDriver' on port 46683.
23/08/21 22:31:13 INFO SparkEnv: Registering MapOutputTracker
23/08/21 22:31:13 INFO SparkEnv: Registering BlockManagerMaster
23/08/21 22:31:13 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
23/08/21 22:31:13 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
23/08/21 22:31:13 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
23/08/21 22:31:13 INFO DiskBlockManager: Created local directory at /mnt/tmp/blockmgr-1a0a2e7a-1643-4325-b208-6860891f183d
23/08/21 22:31:13 INFO MemoryStore: MemoryStore started with capacity 912.3 MiB
23/08/21 22:31:13 INFO SparkEnv: Registering OutputCommitCoordinator
23/08/21 22:31:13 INFO SubResultCacheManager: Sub-result caches are disabled.
23/08/21 22:31:13 INFO JettyUtils: Start Jetty 0.0.0.0:4040 for SparkUI
23/08/21 22:31:13 INFO Utils: Successfully started service 'SparkUI' on port 4040.
```

Fig. 9. Spark running

Step 7: View History of Spark Application

You can monitor the progress and status of your Spark application through the Spark Web UI. - On your cluster's page in AWS, under 'Cluster management' click on 'YARN timeline server'. This will open a Hadoop page with a history of all your spawned applications.

- Click on 'History' for your application, and it will show you a detailed job history that took place during the script, including the stages of when they took place, how long, and how much data was used.



Spark Jobs (?)

User: hadoop
Total Uptime: 1.1 min
Scheduling Mode: FIFO
Completed Jobs: 27

▸ Event Timeline
▾ Completed Jobs (27)

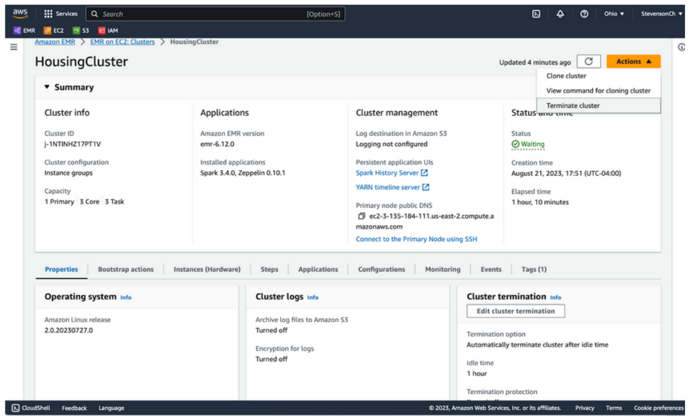
Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
26	treeAggregate at Statistics.scala:58 treeAggregate at Statistics.scala:58	2023/08/21 22:38:40	0.9 s	1/1	1/1
25	treeAggregate at Statistics.scala:58 treeAggregate at Statistics.scala:58	2023/08/21 22:38:39	0.6 s	1/1	1/1
24	collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663	2023/08/21 22:38:39	0.3 s	2/2	2/2
23	collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663	2023/08/21 22:38:38	0.3 s	2/2	2/2
22	collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663	2023/08/21 22:38:38	0.2 s	2/2	2/2
21	collectAsMap at RandomForest.scala:663 collectAsMap at RandomForest.scala:663	2023/08/21 22:38:38	0.2 s	2/2	2/2
20	collectAsMap at RandomForest.scala:663	2023/08/21 22:38:37	1.0 s	2/2	2/2

Fig. 10. Spark Page of all the jobs that took place for the CaliforniaHousing application

Step 8: Completion and Clean-Up

Once your Spark application has completed its execution, review the output logs for any results or errors. Remember to terminate your EC2 instance to avoid incurring unnecessary charges and delete anything from your S3 bucket.



HousingCluster

▾ **Summary**

Cluster info
Cluster ID: j-1NTINHZ17PT1V
Cluster configuration: Instance groups
Capacity: 1 Primary | 3 Core | 3 Task

Applications
Amazon EMR version: emr-6.12.0
Installed applications: Spark 3.4.0, Zeppelin 0.10.1

Cluster management
Log destination in Amazon S3: Logging not configured
Persistent application UIs: Spark History Server, YARN timeline server
Primary node public DNS: ec2-3-135-184-111.us-east-2.compute.amazonaws.com
Connect to the Primary Node using SSH

Status and time
Status: Waiting
Creation time: August 21, 2023, 17:51 (UTC-04:00)
Elapsed time: 1 hour, 10 minutes

Operating system
Amazon Linux release: 2.0.20230727.0

Cluster logs
Archive log files to Amazon S3: Turned off
Encryption for logs: Turned off

Cluster termination
Termination option: Automatically terminate cluster after idle time
Idle time: 1 hour
Termination protection

Fig. 11. Screenshot showing the EC2 dashboard with the 'Terminate instance' option highlighted

IV. Data Visualization

Before training our regression models on the California Housing Data Set, we visualized our data set to see if we could manually find any trends in the data set [1]. As we are using the data set to predict housing prices the column for housing prices is the column we are most interested in. Based on the correlation rates in the column for median house value, it is evident that median income has the highest positive correlation with median house value with a correlation rate of 77%. The next highest correlation rate for a feature in the data set is ocean proximity with a correlation rate of 27%.

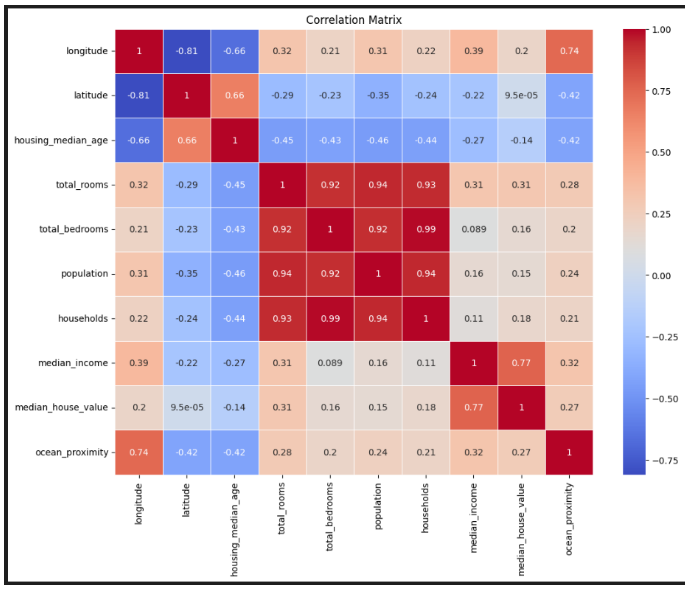


Fig. 12. Correlation Matrix

The high positive correlation between median income and median house value is highlighted in the scatter plot below. The strong positive relationship between the median income and house value will have a strong effect on our training models.

V. Regression Modeling Evaluation

In running a range of regression models on the California Housing Data Set, we obtained the following results for Linear Regression, Decision Tree Regression, and Random Forest Regression. The Performance of each training model is summarized in R-squared and Root Mean Squared Error [1].

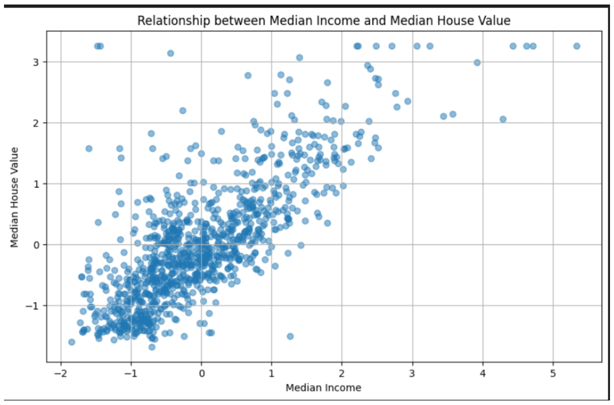
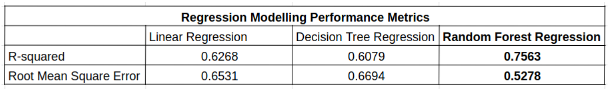


Fig. 13. Scatter plot: relation between median income and median house value

R-squared measures the proportion of the variance in the output variable based on the input variables. A perfect fit between the feature set and the label set would yield a value of 1. With values ranging from 0 to 1, R-squared communicates the amount of noise between each feature used to predict the label. An analysis of the results of our training models resulted in Random Forest Regression having the highest R-square value of 75.64%, while Linear Regression and Decision Tree Regression had similar R-squared statistics of 62.68% and 62.81%, respectively.



Regression Modelling Performance Metrics

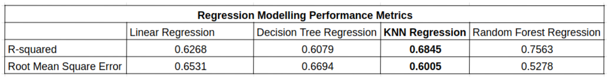
	Linear Regression	Decision Tree Regression	**Random Forest Regression**
R-squared	0.6268	0.6079	**0.7563**
Root Mean Square Error	0.6531	0.6694	**0.5278**

Fig. 14. Performance Metrics

In addition to R-squared we also analyzed performance through Root Mean Squared Error. The classic metric used in gradient descent for regression modeling, Root Mean Squared Error is a loss function that measures the difference between predicted and actual values passed to the regression model. Given that a lower Root Mean Squared Error value is optimal indicating a lesser cost between the prediction and actual value, Random Forest Regression not only offers the best fit between the feature set and label set but is also the most accurate with a Root Mean Squared Error of 0.5278. This is significantly more accurate than the Root Mean Squared Error of Linear Regression and Decision Tree Regression which have error rates of 0.6531 and 0.6519 units, respectively. Overall, the Random Forest Regression is the optimal regression model out of the set of regression models used on the California Housing Data Set. The combination of a relatively high R-squared value and a low Root Mean Squared Error suggests greater accuracy in its predictive powers.

VI. Future Improvements

For our project, we implemented three supervised learning algorithms: Linear Regression, Decision Tree Regression, and Random Forest Regression. To run further experiments we could also expand our training models to include an unsupervised learning algorithm, such as KNN Regression to derive further insights from our data sets. As Decision Tree Regression and Linear Regression perform similarly a trend in the data cannot be ascertained between the two training models. In fact, in running the KNN Regression outside of our AWS cluster we were able to obtain the following results for running a KNN Regression Model on the California Housing Data Set used in our initial three supervised learning trials.



Regression Modelling Performance Metrics				
	Linear Regression	Decision Tree Regression	**KNN Regression**	Random Forest Regression
R-squared	0.6268	0.6079	**0.6845**	0.7563
Root Mean Square Error	0.6531	0.6694	**0.6005**	0.5278

Fig. 15. KNN Regression Metrics

From the figure above it is evident that KNN Regression is second to Random Forest Regression in both R-squared and Root Mean Square Error metrics, with a 68.45% R-squared and 60.05% Root Mean Square Error Rate. The predictive qualities based on the feature set are second highest out of all the regression models used according to the R-square metric, while the error rate captured in the Root Mean Square Error metric is the second lowest. In both instances, KNN Regression comes in second to Random Forest Regression in terms of performance indicating a relatively high accuracy in comparison to the rest of the regression models implemented in the project.

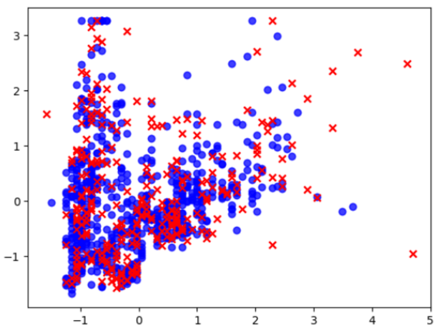


Fig. 16. KNN Regression Scatter Plot

From the strong performance results of KNN Regression and the associated KNN Regression Scatter Plot (Fig. 16), it is evident that data localization occurs within the data set, as certain home owner's with similar income levels tend to buy homes with similar housing prices in the state of California. Given the unsupervised nature of KNN Regression it is also important to confirm our findings with a domain expert. Due to the limitations of the MLlib Spark API, our project did not include the KNN Regression Model when connecting to our AWS cluster. A future iteration of the project could integrate Spark's latest API in order to incorporate a KNN Regression Model with the set of regression models already deployed to a production environment.

APPENDIX A
REGRESSION MODELLING CODE

```python
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler, VectorAssembler, Imputer, StringIndexer
from pyspark.ml.regression import LinearRegression, RandomForestRegressor, DecisionTreeRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
import pyspark.sql.functions as F
from itertools import chain

# Create Spark session
spark = SparkSession.builder.appName("housing").getOrCreate()

# Load data from AWS S3
housing_df = spark.read.csv("s3://awsbucket-achen33/housing.csv", header=True, inferSchema=True)

# Assuming the following mapping for 'ocean_proximity' (you can change this accordingly)
myMapCat = {
    'NEAR BAY': 1,
    'INLAND': 2,
    '<1H OCEAN': 3,
    'NEAR OCEAN': 4,
    'ISLAND': 5
}

map_expr = F.create_map([F.lit(x) for x in chain(*myMapCat.items())])
housing_df = housing_df.withColumn("ocean_proximity", map_expr[housing_df["ocean_proximity"]])

# Handle Missing Values
imputer = Imputer(strategy="median", inputCols=["total_bedrooms"], outputCols=["total_bedrooms"])
housing_df = imputer.fit(housing_df).transform(housing_df)

# Feature Scaling
features = [col_name for col_name in housing_df.columns if col_name != "median_house_value" and col_name != "ocean_proximity"]
assembler = VectorAssembler(inputCols=features, outputCol="features")
scaler = StandardScaler(inputCol="features", outputCol="scaled_features")

# Train-Test Split
train_data, test_data = housing_df.randomSplit([0.7, 0.3], seed=0)

# Linear Regression
lin_reg = LinearRegression(featuresCol="scaled_features", labelCol="median_house_value")
pipeline_lin = Pipeline(stages=[assembler, scaler, lin_reg])
model_lin = pipeline_lin.fit(train_data)
predictions_lin = model_lin.transform(test_data)

# Decision Tree Regression
dt_reg = DecisionTreeRegressor(featuresCol="scaled_features", labelCol="median_house_value")
pipeline_dt = Pipeline(stages=[assembler, scaler, dt_reg])
model_dt = pipeline_dt.fit(train_data)
predictions_dt = model_dt.transform(test_data)

# RandomForest
rf_reg = RandomForestRegressor(featuresCol="scaled_features", labelCol="median_house_value", numTrees=100)
pipeline_rf = Pipeline(stages=[assembler, scaler, rf_reg])
model_rf = pipeline_rf.fit(train_data)
predictions_rf = model_rf.transform(test_data)

evaluator = RegressionEvaluator(labelCol="median_house_value", predictionCol="prediction")

# Evaluate Linear Regression
r2_lin = evaluator.evaluate(predictions_lin, {evaluator.metricName: "r2"})
rmse_lin = evaluator.evaluate(predictions_lin, {evaluator.metricName: "rmse"})
print("Linear Regression - R-squared:", r2_lin)
print("Linear Regression - Root Mean Squared Error:"
    , rmse_lin)

# Evaluate Decision Tree
r2_dt = evaluator.evaluate(predictions_dt, {
    evaluator.metricName: "r2"})
rmse_dt = evaluator.evaluate(predictions_dt, {
    evaluator.metricName: "rmse"})
print("Decision Tree - R-squared:", r2_dt)
print("Decision Tree - Root Mean Squared Error:",
    rmse_dt)

# Evaluate Random Forest
r2_rf = evaluator.evaluate(predictions_rf, {
    evaluator.metricName: "r2"})
rmse_rf = evaluator.evaluate(predictions_rf, {
    evaluator.metricName: "rmse"})
print("Random Forest - R-squared:", r2_rf)
print("Random Forest - Root Mean Squared Error:",
    rmse_rf)

# Close Spark session
spark.stop()
```

References

[1] Funmilola Alaba Ajala Ololade Oyewo Yetunde Faith Akande Gbenle Oluwadara Abigail Bola Adetunji, Oluwatobi Noah Akande. House price prediction using random forest machine learning technique. *Procedia Computer Science*, pages 806–813, 2022.

[2] Cam Nugent. California housing prices.

[3] Durai Raj Vincent Dharmendra Singh Rajput Shaik Abdul Khalandar Basha, Syed Muzamil Basha. Chapter 11 - challenges in storing and processing big data using hadoop and spark. *Academic Press*, pages 179–187, 2019.