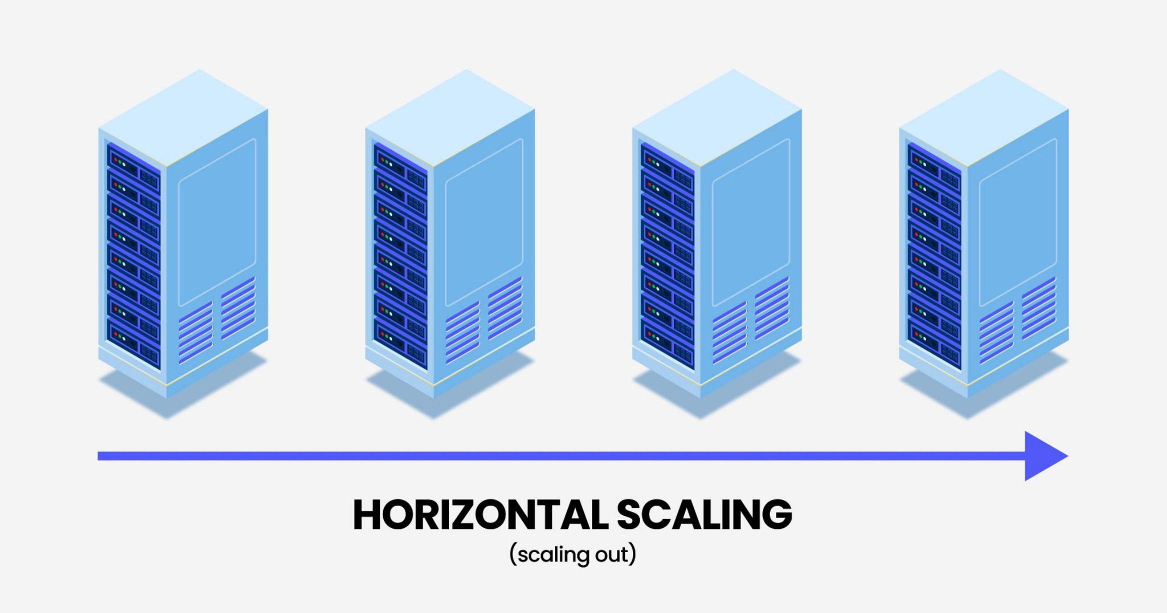
532 Final Project

Anish Mitagar, Karishma Manchanda, Scott Fortune, Sugun Yadla

Project Description

Our project aims to discover the effects of horizontal scaling on the execution time of a basic ML pipeline. We measure the execution time of each stage in the pipeline.

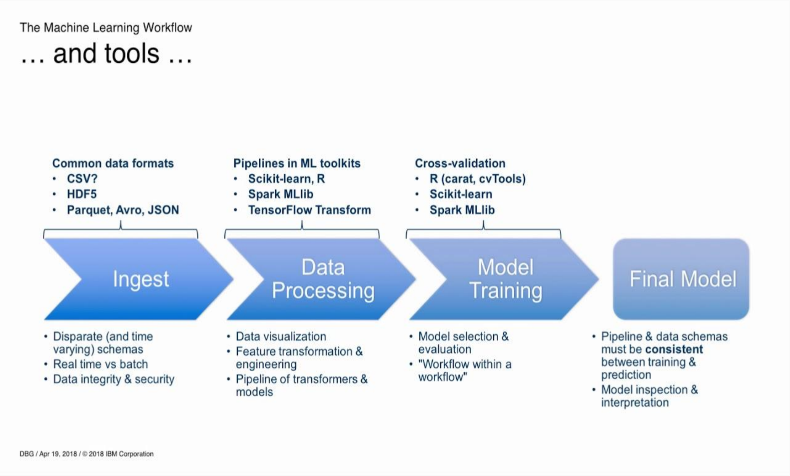


Dataset: Diabetes Diagnosis

- Diabetes prediction dataset: collection of medical and demographic data from patients, plus their diabetes diagnosis (positive or negative)
- The dataset has many attributes that justify different types of data processing/transformation like One-Hot Encoding and Normalization
- Link: https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset

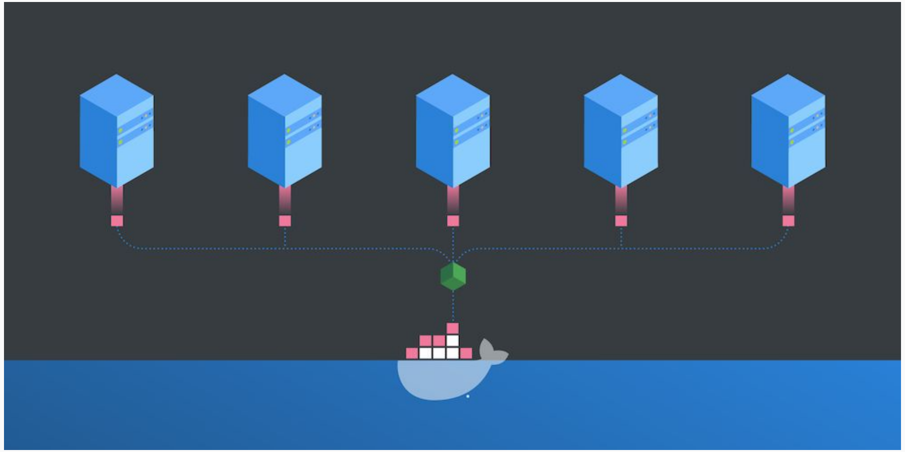
Design Description

- Implemented a pipeline using Apache Spark that predict diabetes in patients based on their medical history and demographic information



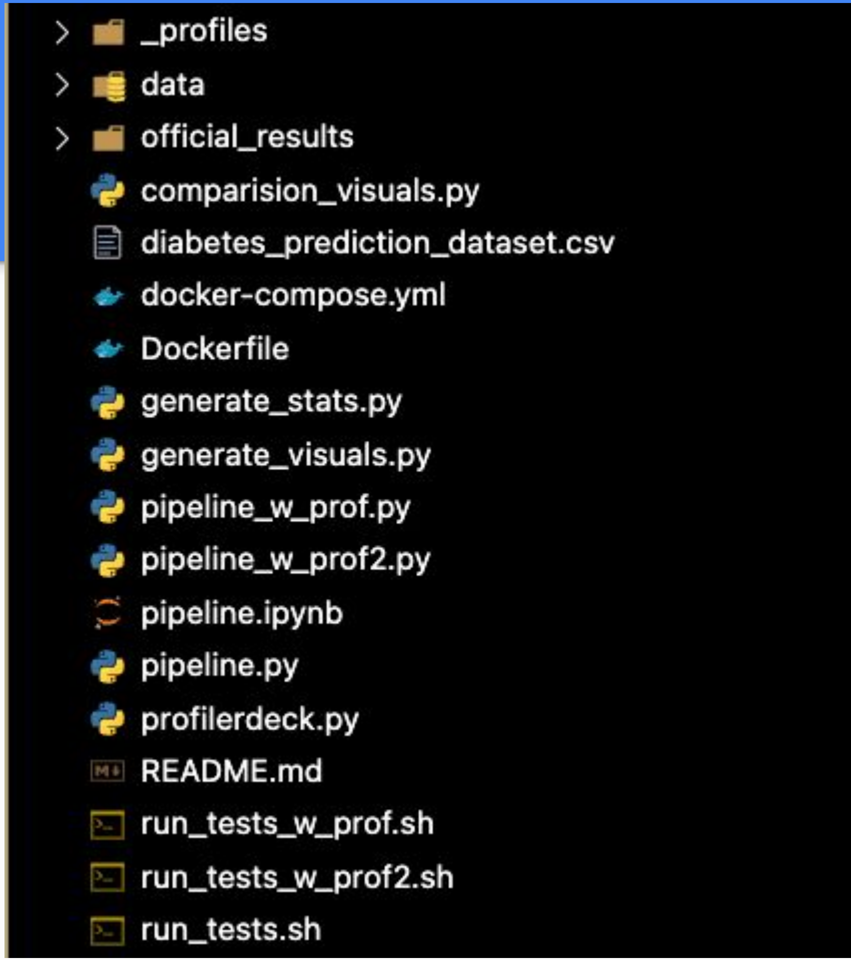
Design Description

- Using Docker + Docker Compose to start up multiple containers (or small virtual machines) within one host machine to best simulate a distributed computing environment



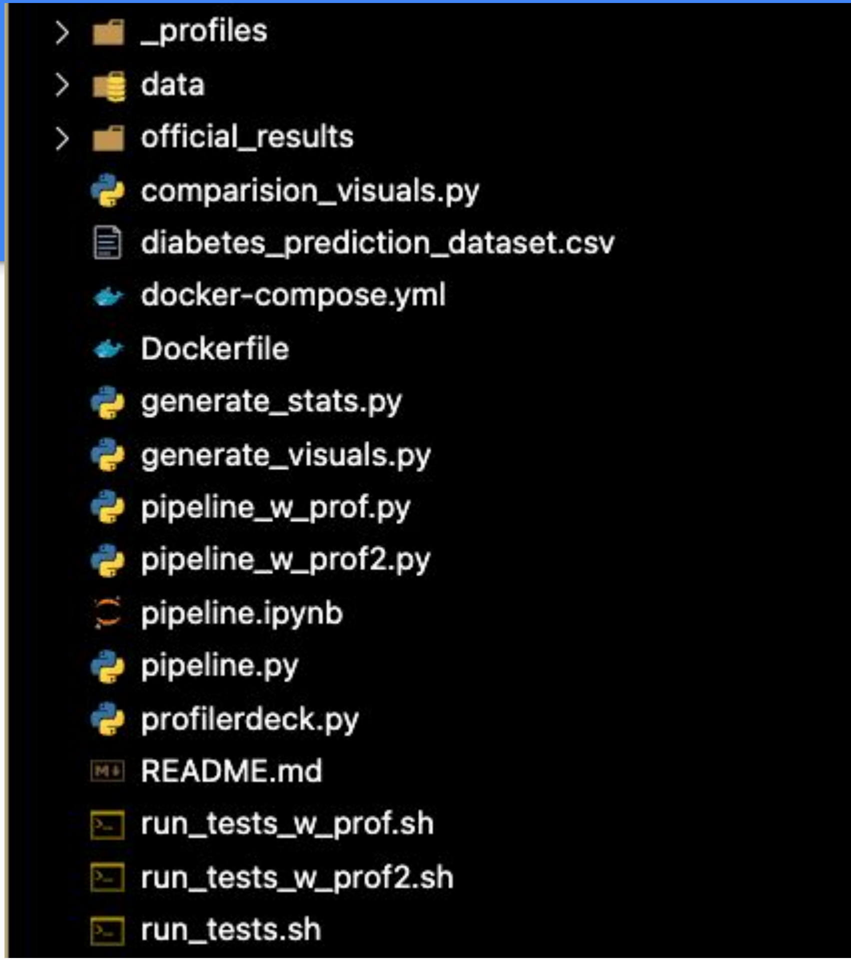
Program Description

- data: stores execution times recorded in the experiments
- diabetes_prediction_dataset.csv: our dataset for training random forests
- docker-compose.yml: defines and configures all services, networks and volumes for a multi-container Docker application
- Dockerfile: Defines and generates docker images
- _profiles: stores the execution profiles of functions called in pipeline.py (and the other variants)



Program Description

- run_tests.sh: a script for running the experiments
- generate_visuals.py: generates graphs of the execution time distributions of the pipeline stages per experiment run
- comparision_visuals.py: generate graphs comparing the execution time distributions for two experiments
- generate _stats.py: generate basic statistics for the execution times of the pipeline stages per experiment



Design Decisions

- PySpark over SciKit Learn: We had experience using PySpark from earlier homeworks; more support for distribution.
- Docker or Kubernetes: Team members had some exposure with Docker. Docker-compose was an easy solution for setting up many containers to run together.
- Trouble running the pipeline on Windows OS versus a Unix based OS (Experiments were run on Macbook Pro M1 Max 32GB RAM)

Tests

- run_tests.sh: Runs the pipeline.py on multiple hardware configurations, confirming that the pipeline executes correctly in different hardware configurations. By Hardware configurations we mean number of running Docker container nodes/workers
- pipeline.py: We test the ML pipeline and the printed accuracy of the chosen best model on predicting the test set. The accuracy was consistently around 97%

More Tests: Using Profiler

```python
@profile
def benchmark_preprocessing_stage(stage, df):
    start_time = time.perf_counter()
    # For estimators (like StringIndexer, OneH
    if isinstance(stage, StringIndexer) or isi
        model = stage.fit(df)
        df = model.transform(df)
    else:  # For transformers (like VectorAsse
        df = stage.transform(df)
    end_time = time.perf_counter()
    return end_time - start_time, df

@profile
def benchmark_validator_stage(crossval, df):
    start_time = time.perf_counter()
    cvModel = crossval.fit(df)
    end_time = time.perf_counter()
    return end_time - start_time, cvModel
```

versus

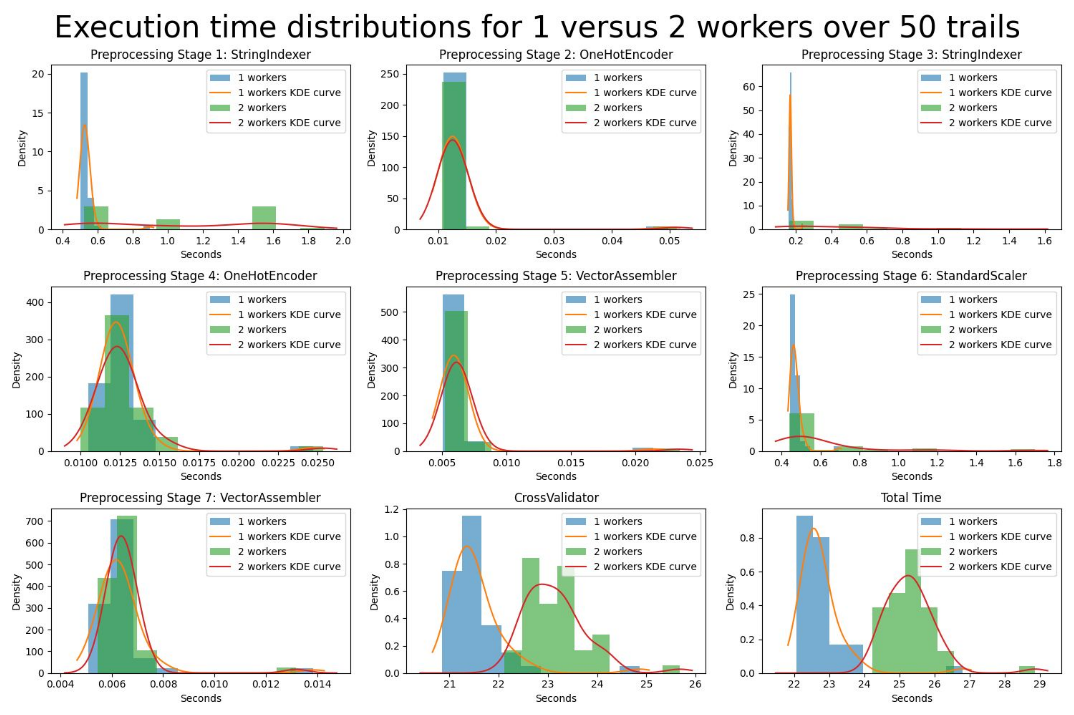
```python
@profile
def benchmark_preprocessing_stage(stage, df):
    # For estimators (like StringIndexer, OneH
    if isinstance(stage, StringIndexer) or isi
        model = stage.fit(df)
        df = model.transform(df)
    else:  # For transformers (like VectorAsse
        df = stage.transform(df)
    return df

@profile
def benchmark_validator_stage(crossval, df):
    cvModel = crossval.fit(df)
    return cvModel
```

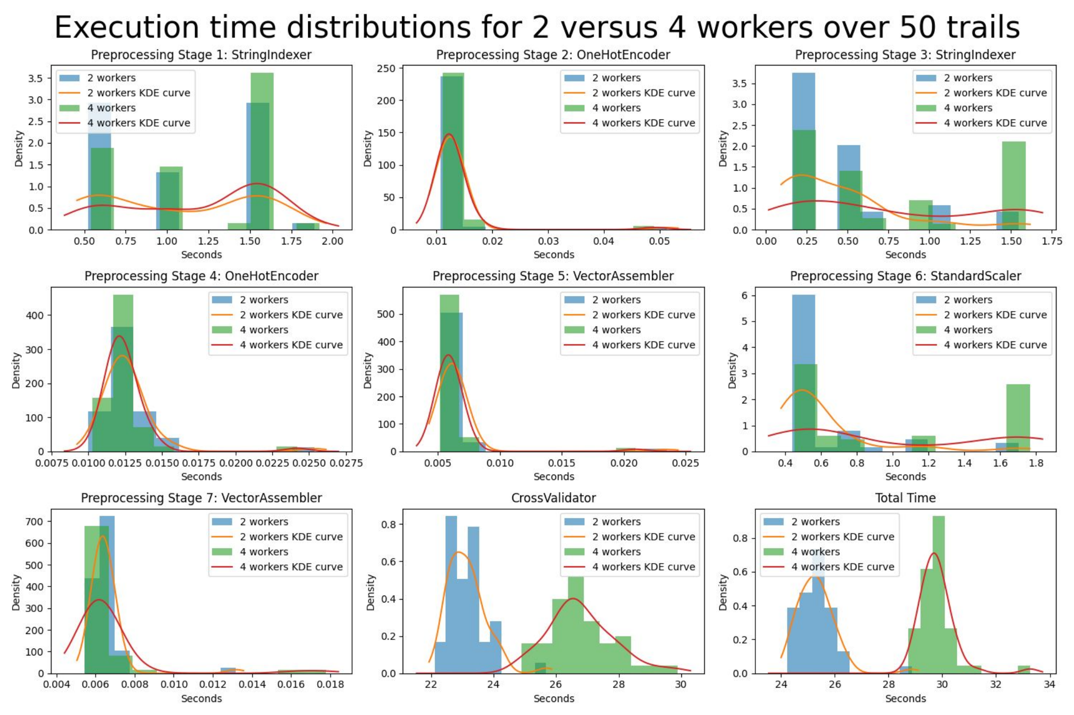
```python
for stage in stages:
    start_time = time.perf_counter()
    df = benchmark_preprocessing_stage(stage, df)
    end_time = time.perf_counter()
    time_taken = end_time - start_time
```

```python
start_time = time.perf_counter()
cvModel = benchmark_validator_stage(crossval, df)
end_time = time.perf_counter()
time_taken = end_time - start_time
```

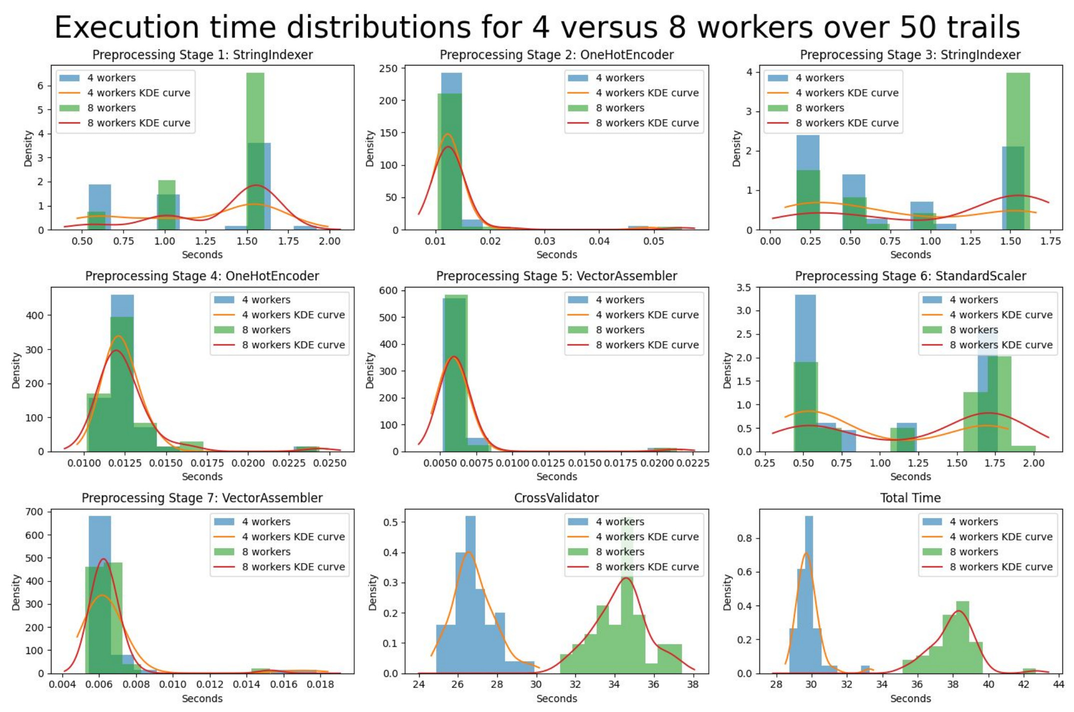
Results and Analysis



Results and Analysis



Results and Analysis



Results and Analysis

	Preprocessing S	Preprocessing S	Preprocessing S	Preprocessing S	Preprocessing S	Preprocessing S	Preprocessing S	CrossValidator	Total Time
Results_num_node_1_Mean	0.5365745019	0.01327376996	0.1703237659	0.01256727912	0.00623423502	0.4716166877	0.00638390164	21.53970804	22.75668218
Results_num_node_1_Median	0.528660334	0.012477271	0.1686474165	0.012259667	0.005924146	0.463601938	0.006223312	21.39982236	22.61244868
Results_num_node_1_Standard Deviation	0.05459817772	0.005353088124	0.01114089496	0.001923141993	0.002240763984	0.03681296633	0.001217014795	0.6143254101	0.7053366261
Results_num_node_2_Mean	1.062638651	0.01329799162	0.4539008586	0.0126862509	0.0065000842	0.6098502513	0.00652169742	23.15251854	25.31791432
Results_num_node_2_Median	1.01245698	0.0124151665	0.244146396	0.0124145835	0.006139854	0.4918484375	0.006338146	23.08982899	25.2816151
Results_num_node_2_Standard Deviation	0.4605166768	0.005565021228	0.3766766478	0.002142188726	0.002489520691	0.2882362228	0.001039239486	0.6258136784	0.7482981964
Results_num_node_3_Mean	1.097853071	0.01326023334	0.9262228497	0.01237177504	0.00641299334	0.677088747	0.00649725254	24.72184479	27.46155171
Results_num_node_3_Median	1.017930084	0.0119575625	1.002200709	0.0119813125	0.0060433745	0.5091217505	0.00629575	24.71241541	27.32626251
Results_num_node_3_Standard Deviation	0.4248585374	0.005624571661	0.5646432137	0.002367143496	0.002819103594	0.3809354147	0.001336245401	0.9200762802	0.7633365385
Results_num_node_4_Mean	1.195033335	0.0132309866	0.7850728179	0.01250561752	0.00619945906	0.990270543	0.00667524172	26.79327042	29.80225842
Results_num_node_4_Median	1.518952897	0.012214271	0.5731200205	0.0121405835	0.0057523745	0.709190834	0.006181958	26.63037978	29.71743664
Results_num_node_4_Standard Deviation	0.4314810406	0.00532045113	0.5745930187	0.001880472807	0.002183372108	0.5539303891	0.002209242173	1.058287846	0.6923572252
Results_num_node_5_Mean	1.261407485	0.01312077662	0.8971519637	0.01239413078	0.00641394584	1.189646516	0.00636872826	28.59252084	31.97902438
Results_num_node_5_Median	1.537422646	0.0123151045	1.001204396	0.0121657915	0.005961875	1.181402126	0.006124042	28.44641224	31.9047477
Results_num_node_5_Standard Deviation	0.4060235487	0.005577354027	0.5832247807	0.00199654637	0.002223406336	0.5451589236	0.001220676125	1.03791027	0.8152981632
Results_num_node_6_Mean	1.216500569	0.0134856158	1.031333715	0.01275883334	0.00631659664	1.237027096	0.00647354082	30.5106642	34.03456017
Results_num_node_6_Median	1.523853376	0.0127132295	1.520886064	0.012277604	0.006002833	1.678838834	0.006181021	30.51015966	34.02012881
Results_num_node_6_Standard Deviation	0.4351735279	0.005598707519	0.5921442511	0.00202098705	0.002113930497	0.5401002437	0.001415681324	1.217653118	0.9786246947
Results_num_node_7_Mean	1.321246313	0.01392664412	1.203815786	0.01312679342	0.00639042998	1.30879657	0.00641633504	32.18570125	36.05942012
Results_num_node_7_Median	1.541004271	0.0130031455	1.546808584	0.0120884165	0.005995271	1.686779333	0.006220729	32.24763445	36.12089064
Results_num_node_7_Standard Deviation	0.3817497358	0.005748684052	0.4961197265	0.003297873141	0.002090911388	0.5188681851	0.00133704967	1.123426008	0.9038165686
Results_num_node_8_Mean	1.356419476	0.01348154668	1.088352522	0.0125400766	0.00632114758	1.228671177	0.0064810192	34.30915959	38.02142656
Results_num_node_8_Median	1.542367375	0.0122324375	1.532600959	0.011981292	0.005925770999	1.67522548	0.0063131045	34.54633891	38.11362987
Results_num_node_8_Standard Deviation	0.3274429982	0.006257810329	0.5895134726	0.002078609284	0.002229696371	0.5703185752	0.001363003769	1.341448729	1.228832134

Results and Analysis

- Expectation: greater number of nodes/workers → runtime decreases
- Result: greater number of nodes/workers → runtime increased
- Two Possible Explanations:
- Communication Overhead between nodes/workers > processing gains
- Overhead from nodes/workers competing for limited resources
    - Processes > Cores available per node/worker

Goals

- Runtime on Distributed vs Single System (Success with caveat): we only had the resources and time to conduct the research on a single system, but we did conduct research on multiple “virtual” machines on a single system
- Runtime vs Number of Nodes (Success): Created graphs and conducted analysis for 1, 2, 4, and 8 running workers/nodes
- Demonstrated the benefits of distributed computing on heavy workloads (Failed): insufficient hardware setup because of lack of time and money

Possible Improvements

- Better hardware: Poor execution times most likely due to running ML code on laptops. Overhead of distribution protocols may be more prominent on lower spec hardware.
- Explore more ML Families: Performances gains scaling when other model families are trained such as neural networks.
- Pick more appropriate workload for appropriate hardware: Perhaps ML pipeline is not meant for lower spec hardware.
- Better simulation of distributed computing (on a budget): Instead of a single host machine with many small VMs, link several physical Raspberry Pis together