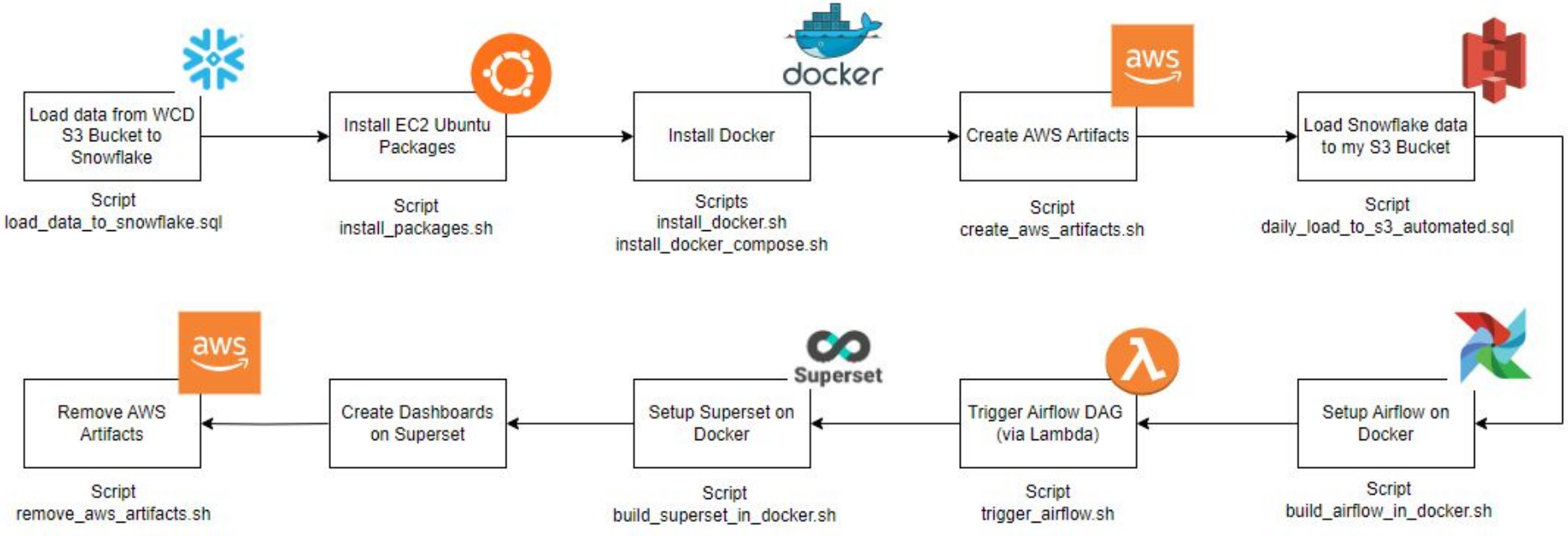
WeCloud Data Engineering Midterm Presentation

Nathan Ling
EST Batch 7

User Journey

Minimize the number of user clicks



Exploratory Data Analysis

- Created a Databricks file and bucket mount for analysis
  - Save costs by using the free community Databricks edition
- Earliest date is Jan 1, 2020 and most recent date is today
- For every row in the inventory table, there exists a row in the sales table
  - However, the opposite is not true!
- The dataset only consists of American stores

EC2 Ubuntu Installations

- Prevent errors downstream
- Minimize the pain of setting up the packages, which include
  - Zip
  - Python 3.7
  - AWS CLI
  - Pip
  - Boto3
  - Dotenv
  - Toml
  - Docker
  - Docker Compose

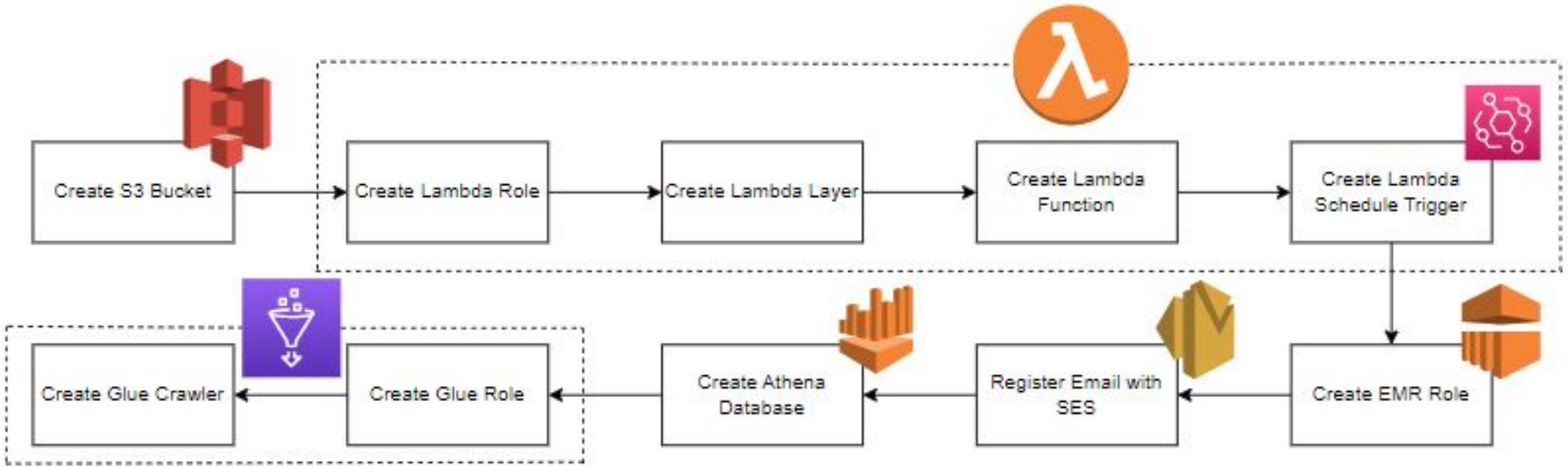
```
#!/bin/bash

# Please replace apt-get with yum if you are using Amazon AMI, which uses redhat
sudo apt-get update
sudo apt-get install python3.7 -y
sudo apt-get install awscli -y
sudo apt-get install pip -y
sudo apt-get install zip -y
pip3 install boto3
pip3 install python_dotenv
pip3 install toml
```

Config Files

1. Ensures all the code is referencing the same source of truth
2. Minimizes user work
    a. Will be tedious to change each .sh or .py file to manually edit variables

```
[aws]
s3_bucket_input_and_script='input-bucket-de-midterm-nl'
s3_bucket_output='output-bucket-de-midterm-nl'
region='ca-central-1'
lambda_function_name='de_midterm_lambda_function_automated'
```

Create AWS Artifacts



Create AWS Artifacts Cont.

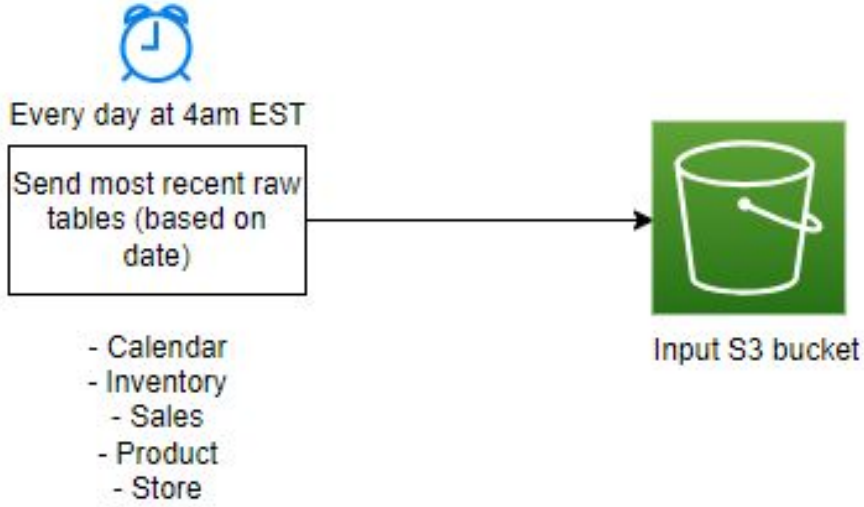
- Keeping the code modular
- Enabling user control

```
read -p "Have you created a Lambda function yet? [y/n] " continue
case $continue in
    "n"|"no"|"N"|"NO"|"No"|"nO")
    echo "Creating the Lambda function"
    sleep 3
    chmod +x create_lambda_function.sh
    ./create_lambda_function.sh
    RC1=$?
    if [ $RC1 != 0 ]; then
        echo "Failed to create a Lambda function"
        echo "[ERROR:] RETURN CODE:  $RC1"
        echo "[ERROR:] REFER TO THE LOG FOR THE REASON FOR THE FAILURE."
        exit 1
    else
        echo "Lambda function created"
    fi
;;
esac
```

```
# Zipping all code (includes the python script, config toml, and .env files)
zip lambda_code.zip lambda_function.py ../config_file.toml ../.env

# Push code package to S3
aws s3 cp lambda_code.zip s3://${input_s3_bucket}/scripts/lambda_code.zip

# Create lambda function
aws lambda create-function \
 --function-name ${function_name} \
 --region ${region} \
 --code S3Bucket=${input_s3_bucket},S3Key="scripts/lambda_code.zip" \
 --handler lambda_function.lambda_handler \
 --runtime python3.7 \
 --role arn:aws:iam::${account_id}:role/${lambda_role_name} \
 --layers arn:aws:lambda:${region}:${account_id}:layer:${lambda_layer_name}:${lambda_layer_version} \
 --timeout 30

# Removing the zip file to keep the folder clean
rm lambda_code.zip
```

Snowflake

- Mimics OLTP process
- Requires creating
  - Staging area
  - Integration with S3 bucket via AWS role



Lambda Function

1. What it does
   a. Checks if today's data has been uploaded to the input S3 bucket
   b. If so, sends the S3 uri to Airflow
   c. Otherwise, sends an email to the user that today's data is not ready yet
2. Includes .env and config toml file, which requires library packages to be loaded
3. Thus lambda layer is needed

```
# Only activate Airflow if the input bucket has all the required files
if set(required_file_list).issubset(s3_file_list):
    required_file_url = ['s3://' + f'{s3_bucket}/data/' + a for a in required_file_list]
    print('required_file_url: ', required_file_url)
    table_name = [a[:-15] for a in required_file_list]
    print('table_name: ', table_name)
    data = json.dumps({'conf':{a:b for a,b in zip(table_name, required_file_url)}})
    print(data)
# send signal to Airflow
    endpoint=f'http://{ec2_ip_address}:8080/api/v1/dags/{dag_name}/dagRuns'
    user_credentials=f'{os.getenv("AIRFLOW_USERNAME")}:{os.getenv("AIRFLOW_PASSWORD")}'
    subprocess.run([
        'curl',
        '-X',
        'POST',
        endpoint,
        '-H',
        'accept: application/json',
        '-H',
        'Content-Type: application/json',
        '--user',
        user_credentials,
        '--data',
        data])
    print('File are send to Airflow')
else:
    # If required files aren't found in the input bucket, send email to user reminding them of such
    send_email(sender, recipient, aws_region)
```

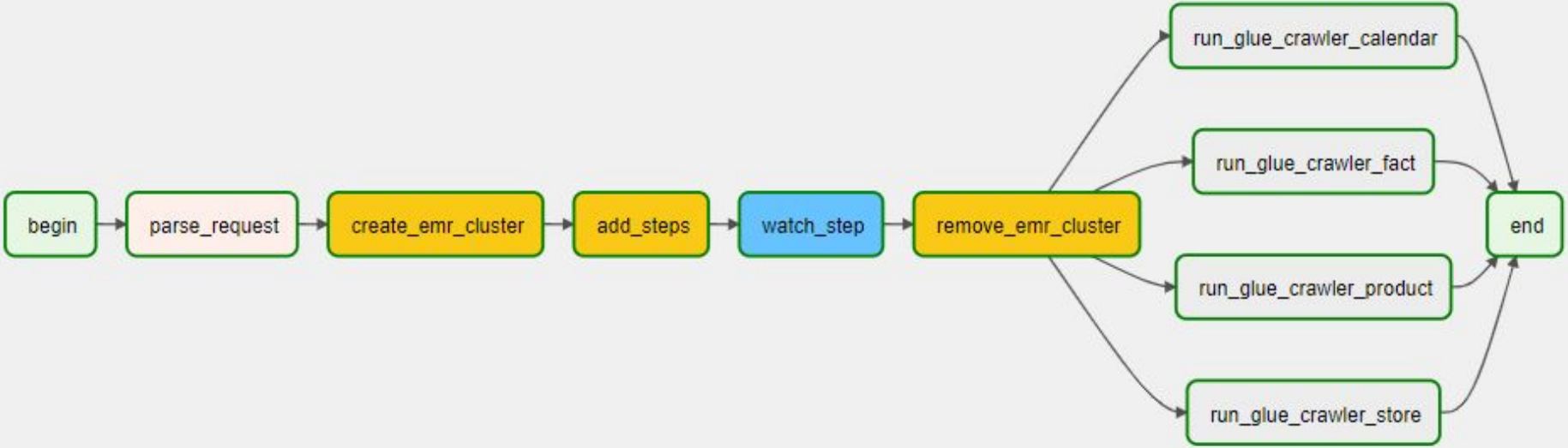
Spark ETL

1. Retrieve input data from Lambda via Airflow API
2. Aggregate sales table by day, product, and store key
3. Join the fact tables (Inventory, Sales) and Calendar table
    a. To get week and day of week columns
4. Create new metrics which include
    a. End of week stock level
    b. Number of out of stock instances
5. Aggregate into fact table grouped by week, product, and store key
6. Push transformed data into S3 output bucket

Spark ETL Cont.

```
# Grouping the sales table into daily aggregations (so that we can join with the inventory table)
daily_sales_df = sales_df.groupBy('TRANS_DT', 'PROD_KEY', 'STORE_KEY')\
    .agg(sum('SALES_QTY').alias('SALES_QTY'),\
    sum('SALES_AMT').alias('SALES_AMT'),\
    sum('SALES_COST').alias('SALES_COST')
    )
daily_sales_df.show(2)

# Joining the inventory, sales, and calendar table
sales_inv_calendar_df = inventory_df\
    .join(daily_sales_df, (inventory_df['CAL_DT']==daily_sales_df['TRANS_DT']) & (inventory_df['PROD_KEY']==daily_sales_df['PROD_KEY']) & (inventory_df['STORE_KEY']==daily_sales_df['STORE_KEY']), 'left')\
    .join(calendar_df, inventory_df['CAL_DT']==calendar_df['CAL_DT'], 'left')\
        .select(inventory_df['CAL_DT'],
                inventory_df['PROD_KEY'],
                inventory_df['STORE_KEY'],
                'SALES_QTY',
                'SALES_AMT',
                'SALES_COST',
                'INVENTORY_ON_HAND_QTY',
                'INVENTORY_ON_ORDER_QTY',
                'OUT_OF_STOCK_FLG',
                'WASTE_QTY',
                'DAY_OF_WK_NUM',
                'YR_WK_NUM'
                )
sales_inv_calendar_df.show(2)
```

Airflow


begin
parse_request
create_emr_cluster
add_steps
watch_step
remove_emr_cluster
run_glue_crawler_calendar
run_glue_crawler_fact
run_glue_crawler_product
run_glue_crawler_store
end

Airflow Cont.

- Run in Docker container
- Additions include sending a config toml file and installing toml library into the image build
- Created the following shell scripts
  - Builds and sets up Airflow
  - Invokes Lambda function which triggers Airflow to start

```
FROM apache/airflow:latest
COPY config_file.toml .
RUN pip install toml
```

```
#!/bin/bash

# Only run this script if either this is your first time building the docker image for Airflow
# or if you are planning to rebuild the image again. Otherwise, there is no need to run this script
# as you will need to set up all the connections and dags again in the Airflow UI.
# Therefore, if you've already built your image and have no intentions of rebuilding it again,
# just run "docker-compose up -d" to start Airflow and "docker-compose down" to shut it down
airflow_name_prefix=$(pwd | rev | cut -d "/" -f 1 | rev | awk '{print tolower($0)}')

# Removing the folders so that there isn't an error if we rebuild the image again
for i in dags pgdata plugins
    do
    if [ -d $i ]; then
        sudo rm -r $i
    fi
    done

# Build docker airflow image and start container
docker-compose build
docker-compose up -d

# Upgrades and initializes the Airflow database
echo "Upgrading airflow database"
docker exec -it ${airflow_name_prefix}_webserver_1 airflow db upgrade

echo "Initializing airflow database"
docker exec -it ${airflow_name_prefix}_webserver_1 airflow db init
RC1=$?
if [ $RC1 != 0 ]; then
    echo "Failed to initialize airflow database"
    echo "[ERROR:] RETURN CODE:  $RC1"
    docker-compose down
    exit 1
fi
```

Superset Dashboards

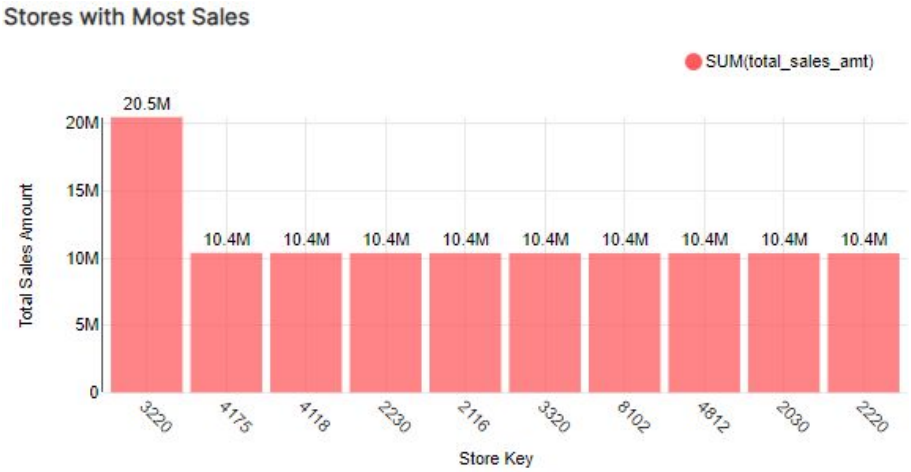
- Created a Superset container through Docker
- Connected to AWS Athena
- Created new joins to extract more features
  - Joined fact table with store and calendar tables to extract geographical and time data
- Filtered out the dashboard to top or bottom 10 for store and product keys
- Dashboard was based on the data as of **July 30, 2023**

Monthly Analysis

- Weekly sales amount fluctuates based on seasons
- However, no stock frequency remains fairly consistent across the year

Store Analysis

- Store 3220 is making the most sales yet is losing the most money
- A strong indicator that the stores have issues with managing expenses

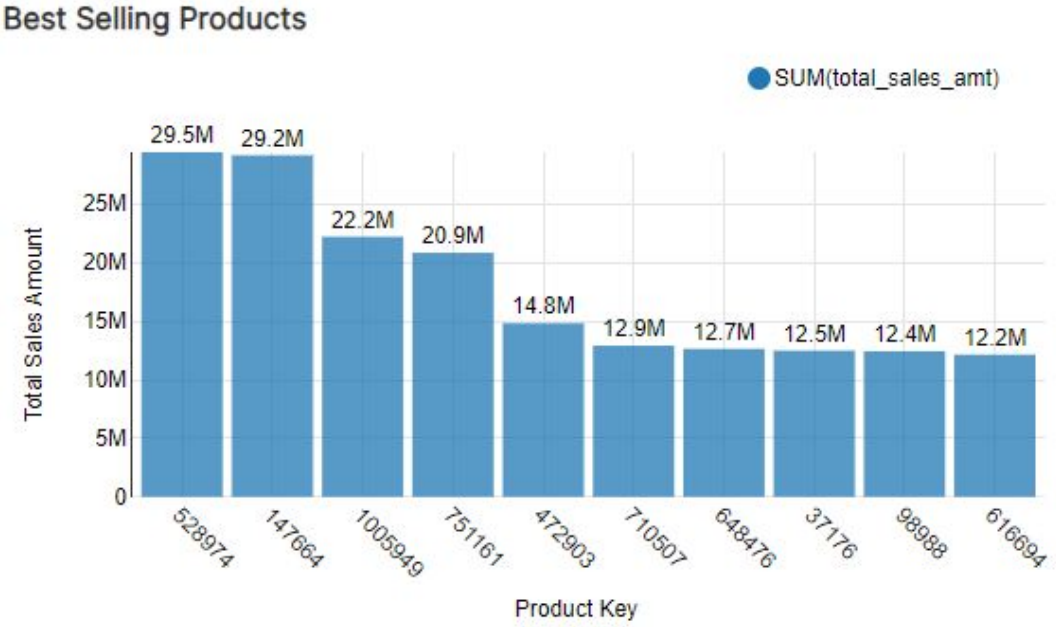


Stores Losing the Most Money

store_key	Total_Loss
3220	-1.97M
3170	-997k
4117	-995k
2345	-991k
2160	-981k
4929	-979k
2020	-973k
2355	-973k
4155	-972k
4880	-970k

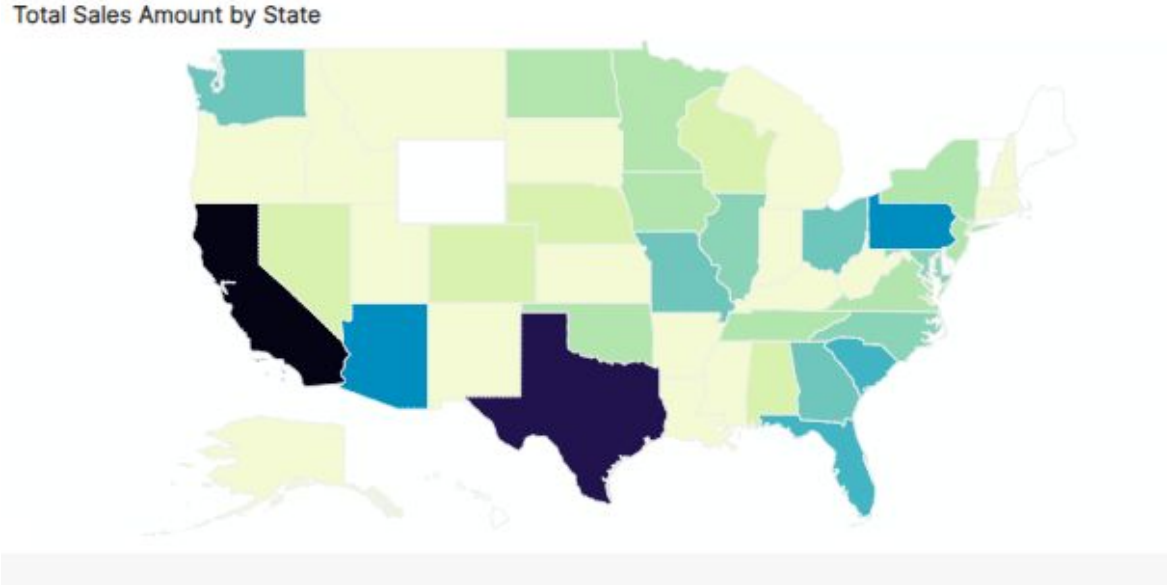
Product Analysis

- Two products lead the way in sales amount, raking in nearly $30M since the beginning of 2020



Geographical Analysis

- California, Texas, and Pennsylvania lead the way in sales amount
  - Likely by virtue of being the most populous states
- Surprisingly, Arizona comes in fourth and New York is middle of the pack

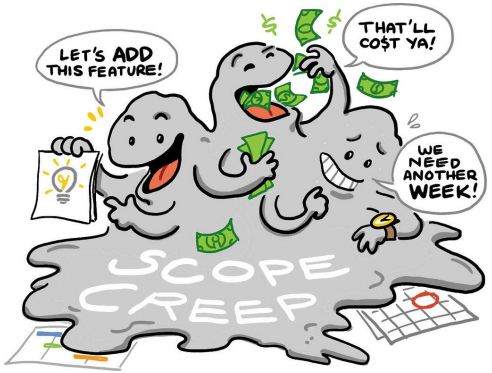


Remove Artifacts

- Removes all the AWS artifacts that were created
- Keeps the AWS account clean
- Shell Script similar to Create AWS Artifacts

```
read -p "Have you removed the Glue crawler yet? [y/n] " continue
case $continue in
    "n"|"no"|"N"|"NO"|"No"|"nO")
    echo "Removing Glue Crawler"
    sleep 3
    chmod +x remove_glue_crawler.sh
    ./remove_glue_crawler.sh
    RC1=$?
    if [ $RC1 != 0 ]; then
        echo "Failed to remove the Glue crawler"
        echo "[ERROR:] RETURN CODE:  $RC1"
        echo "[ERROR:] REFER TO THE LOG FOR THE REASON FOR THE FAILURE."
        exit 1
    else
        echo "Glue crawler removed"
    fi
;;
esac
```

Challenges



1. Automating Data Pipeline
    a. Debugging process was time consuming
    b. Connecting all the steps together
2. Managing Project Scope
    a. It was tempting to keep adding features which would enhance the project
    b. Time constraints put a lid on some of these ambitions
3. Superset
    a. Line graphs were not very feasible due to mismatch in data types
4. Dataset
    a. The Product and Store tables lacked descriptive information

Next Steps

1. Explore IaC (eg. Terraform) to automatically create and destroy EC2 instances
2. Explore Great Expectations
3. Utilize another BI Tool (eg. PowerBI) to develop dashboards
4. Automate the steps in
    a. Airflow UI
    b. Snowflake UI
    c. Superset UI

Thank You