

School of Information Studies
Syracuse University
IST 615 – Cloud Management
Final Project Report
Car Retail Website



Due Date: 12/05/2022

Submission Date: 12/05/2022

Number of Pages: 14

Team members:

Mikhail Pravin Pinto (mpinto01@syr.edu)
Kirat Saran (ksaran@syr.edu)

Contents

S.No.	Description	Page No.
1.	Introduction	2
2.	Implementation	2
3.	Future Scope	14

Introduction:

We are building a car retail website using cloud where buyers can buy cars based on manufacturers, mileage, distance traveled, etc since the demand to buy vehicles has increased post pandemic.

The dataset is from the site Craigslist which was uploaded to Kaggle.com.

Link to the dataset:

<https://www.kaggle.com/datasets/austinreese/craigslist-cartrucks-data?datasetId=62920>

We have used Microsoft Azure Cloud Services to implement the solution.

Implementation:

1. *Data Cleaning*

The dataset was downloaded in the form of a csv file. We cleaned the data by removing rows with null values in columns like condition, manufacturer, model, fuel, odometer, and transmission.

Column with the County name was also dropped as it had more than 50% null values. Similarly, the size and region URL was also removed due to a lot of missing values. The description was also dropped as the column contained identical descriptions for more than 50% of the records.

2. Azure Blob Storage

Our cleaned dataset file ‘vehicles_cleaned.csv’ was uploaded to Azure blob service and stored in a container.

Screenshot 1: A container created in Azure blob storage to upload and store our cleaned csv data file.

The screenshot shows the Azure Blob Storage interface for the container 'ist615team7file'. The left sidebar has 'Overview' selected. The main area shows a table with one row:

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
vehicles_cleaned.csv	11/13/2022, 6:03:11 ...	Hot (Inferred)		Block blob	74.31 MiB	Available

4. Azure SQL Database

The SQL Database was set up so that the data from the blob storage can be integrated with the Database which will finally communicate with the website.

Screenshot 2 and 3: Azure SQL Database setup created for the purpose of data integration.

This screenshot shows the Azure SQL Database Overview page for the 'ist615projectteam7-database'. The left sidebar contains navigation links for Overview, Activity log, Tags, Diagnose and solve problems, Getting started, Query editor (preview), Settings (Compute + storage, Connection strings, Properties, Locks), Data management (Replicas, Sync to other databases), Integrations (Azure Synapse Link, Stream analytics (preview), Add Azure Search), and Power Platform. The main content area displays the database's resource group, status, location, subscription, and server name. It also shows connection strings, pricing tier, auto-pause delay, and earliest restore point. A 'Monitoring' tab is selected, showing key metrics like Compute utilization, App CPU billed, and Database data storage. The Compute utilization chart shows a sharp peak around 2:45 UTC on May 1st. The App CPU billed chart shows a bar chart with a value of 1.91 k. The Database data storage section shows used space (192 MB), allocated space (336 MB), and max storage (32 GB).

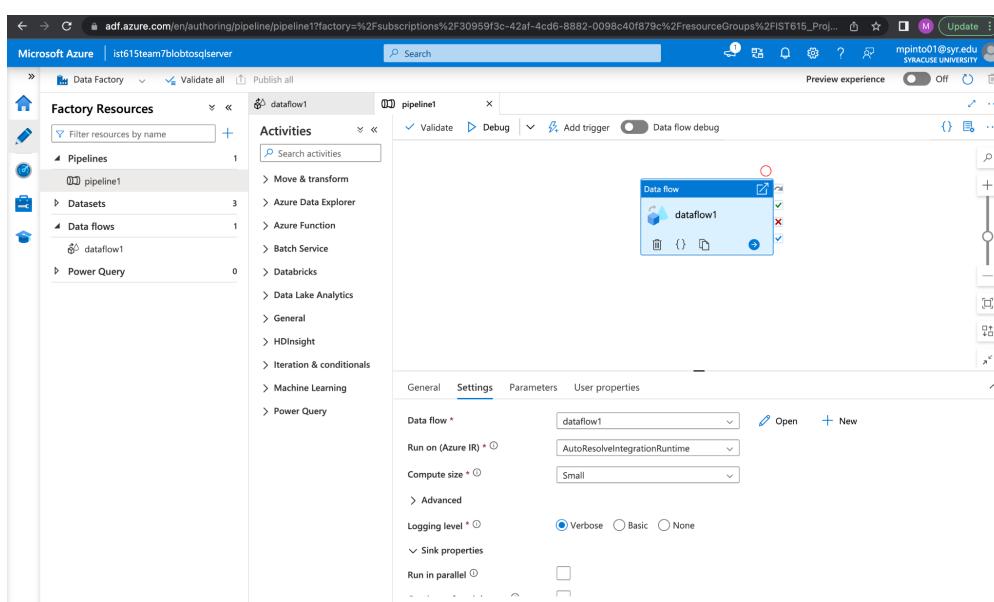
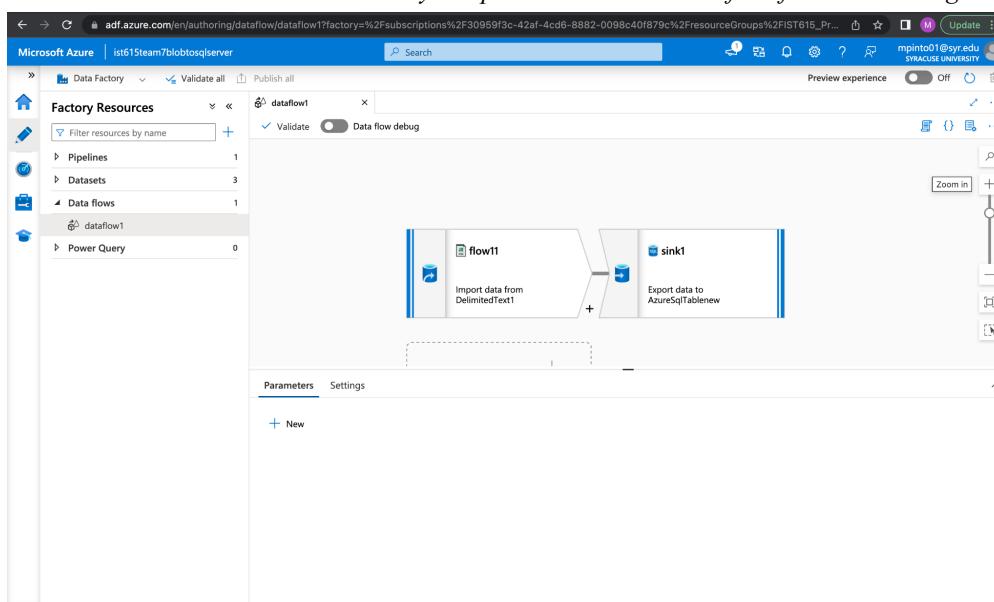
This screenshot shows the Azure SQL Database Monitoring page for the same database. The left sidebar is identical to Screenshot 2. The main content area displays detailed performance metrics over time. The 'Compute utilization' chart shows CPU percentage (Avg) at 0%, Log IO percentage (Avg) at 0%, and Worker percentage (Avg) at 0.0084%. The 'App CPU billed' chart shows the total App CPU billed (Sum) at 1.91 k. The 'Database data storage' section shows the current state of storage usage.

Metric	Value
CPU percentage (Avg)	0%
Log IO percentage (Avg)	0%
Worker percentage (Avg)	0.0084 %
App CPU billed (Sum)	1.91 k

3. Azure Data Factory

Azure Data Factory has been used to facilitate data engineering and data integration. A data flow was created to get data from the Azure Blob storage container to Azure SQL DB. This was executed using a pipeline to get data from Azure blob storage to Azure SQL DB. To achieve this we had to set up the source i.e Azure Blob storage in a right way so that the column headers and the values are accurately analyzed. Finally using the Data flow each field in the csv file from the Azure Blob storage was mapped to the column headers. Using this would make it easy for the destination i.e the Azure SQL Database.

Screenshot 4 and 5: Azure Data Factory setup to create a data workflow from blob storage to SQL DB.



4. Retool Service

ReTool is a platform where they allow their user to connect to databases and create an application which acts like a CRUD application. To create our website we have created three different pages one in which we can look for the cars listed on the website. To do that basic SQL + JavaScript is used as shown in the below screenshot. For retool to communicate with the database some ip's were added to the firewall rules of the database.

Screenshot 6 and 7: SQL queries used for the development of webpage to display cars.

The screenshot shows the ReTool interface for developing a web application to display cars. On the left, there is a search interface with dropdowns for Price Range, Condition, Manufacturer, and Model. The main area displays a table titled "table1" with columns: Price, Manufacturer, Model, Condition, Fuel Type, Odometer, Title Status, Transmission, Type, Paint Color, and Post Date. Below the table, a "TableQuery" configuration panel is visible. It includes a sidebar with filters for TableQuery, Condition, Manufacturer, Model, and PriceRange. The main panel shows the "General" tab with a "Resource" set to "MS SQL Database". Under "SQL mode", it says "Run query automatically when inputs change". The SQL query is:

```

1 select * from dbo.usedcar_listing as a
2 where a.price_bine like {{select4.value}} and
3 a.manufacturer like {{select1.value}} and
4 a.model LIKE {{select2.value}} and
5 a.condition like {{select3.value}}

```

On the right side, the "Inspect" tab is active, showing details about the "table1" table. It lists columns: VIN, condition, cylinders, drive, fuel, id, and image_url, all defined as nvarchar. There are also sections for "Data" (containing the expression `{{ TableQuery.data }}`), "Empty state" (showing "No rows found"), and "Columns" (listing all 23 columns with checkboxes for "Editable").

Each of the cells is attached to a query which gets values from the database based on the selections made by the user. This can be seen in the screenshot below.

The screenshot shows the Retool interface for building a car listing search application. On the left, there's a preview of the user interface with several dropdown filters for Price Range, Condition, Manufacturer, and Model. Below the preview is a table with columns like Price, Manufacturer, Model, Condition, Fuel Type, Odometer, Title Status, Transmission, Type, Paint Color, and Post Date. On the right, the 'Inspect' tab is active, showing the configuration for the 'Model' dropdown. It specifies a data source of 'Model' and a mapped value of '{{ item.model }}'. A note indicates that the selected data source is empty or could not be converted to an array. Below this, there are sections for Label, Caption, Color, Prefix image, Prefix icon, Prefix text, Tooltip, Disabled, and Hidden.

Once the user enters all the fields they will be able to see the listings that are available on the site. It can be seen in the screenshot below. Paint type was not included as we did not have enough information for each category of cars.

Screenshot 8: Webpage for users to list used cars

The screenshot shows the deployed version of the car listing application. The top navigation bar includes 'Car Listings' and 'Latest'. The main page title is 'Used Car Listings'. The search filters are set to '20k to 40k' for Price Range, 'good' for Condition, 'ford' for Manufacturer, and 'coupe' for Model. The search results table shows three rows of data: a Ford coupe for \$22,500, a Ford coupe for \$24,500, and a Ford coupe for \$29,000. To the right of the table is a dropdown menu for 'Model' which lists various car brands. At the bottom left is a Retool logo, and at the bottom right are production and latest status indicators.

If the user wants to check the value of their car, we have created a new page where the user can check the highest and the lowest estimated resale value according to the data that we have in our database. This page can be accessed by clicking the navigation button on the top and selecting know your car's worth. Similar to the first page the user can select options from the select lists that are on the page. Once the user selects all four then they get an estimate of their car. The query used to trigger this action is mentioned in the screenshot below. In that query the highest estimate is the highest value that the car has been listed in our system with i.e by the manufacturer, model, condition and the title status (It tells us the category for times the car has been crashed and the parts have been changed for the same).

Screenshot 9 and 10: SQL queries used for the development of webpage to display price estimates for any user's car

The screenshot shows a web-based application interface for managing car listings. On the left, there's a sidebar with a tree view of tables: 'All tables' (expanded), 'dbo.usedcar_listing' (selected), and its columns: VIN, condition, cylinders, drive, fuel, id, image_url, lat, long, manufacturer, and model. The main area displays a form titled 'Your Car's Resale Estimate' with dropdown menus for Manufacturer, Model, Condition, and Title Status. Below the form, a table header 'Highest Estimate' and 'Lowest Estimate' is shown, followed by a message 'No rows found'. At the bottom, a code editor window shows an SQL query:

```

1 select MAX(price) as 'Highest Estimate', AVG(price) as 'Lowest Estimate'
2 from dbo.usedcar_listing as a
3 where a.manufacturer like {{select1.value}} AND
4 a.model like {{select2.value}} AND
5 a.condition LIKE {{select3.value}} AND
6 a.title_status LIKE {{select4.value}}
6 GROUP BY a.manufacturer, a.model, a.condition, a.title_status

```

The code editor also includes sections for 'General', 'Response', and 'Advanced' settings, and a preview/run button.

Similar to the previous page each selection list is connected to a query which retrieves data from the database. The example for this is in the screenshot below.

The screenshot shows the Retool interface for a "Sell Your Car" application. On the left, there's a preview of the user interface titled "Your Car's Resale Estimate" with four dropdown menus: Manufacturer (Select an option), Model (select4), Condition (Select an option), and Title Status (Select an option). Below the interface, a sidebar shows performance metrics: All queries completed (2.6s for Manufacturer, 2.2s for Model, 2.0s for Condition, 3.7s for TitleStat, 1.2s for Finalop). The main panel displays a SQL query for selecting distinct title statuses from the usedcar_listing table where manufacturer and model match the selected values. To the right, the "Inspect" tab is open, showing the "TitleStat" data source is empty and cannot be converted to an array. The "Mapped" tab shows a mapping for the "Value" field, which is {{ item.title_status }}.

A full working example of this can be seen in the visual below. The highest estimate is the highest ever posted estimate for the selections made by the user and the lowest is the average of all the listings for the selections for that car type. Paint type was not included as we did not have enough information for each category of cars.

Screenshot 11: Webpage for users to list price estimates for their car

The screenshot shows the final web application titled "Your Car's Resale Estimate". The user has selected "ford" as the manufacturer, "model t" as the model, and "good" as the condition. The "Title Status" dropdown is set to "clear". Below the form, a table displays the results: "Highest Estimate" is 16000, and "Lowest Estimate" is 4575. A message at the bottom indicates "Showing 1 result". The Retool logo is visible in the bottom left corner.

Lastly a third page was created so that the users will be able to post their own car for sale. For this the following options were selected. Each of the fields have to be filled in order to post the car for sale. To post the query an Insert action has been used which works exactly like the SQL insert into database command. Each of the fields has been given some constraints to enter the information.

Screenshot 12: Setup to insert the user's car's data that is put for sale into our database.

The screenshot shows the Retool interface for creating a car listing. On the left, there's a form titled "Enter Car Details" with fields for Price, Year, Manufacturer, Model, Condition, Fuel, Odometer, Title status, Transmission, Vin, Drive, Type, and Paint color. Each field has a placeholder and a validation message. On the right, the "Inspect" tab is open for the first input field, "textInput1". The "BASIC" settings include a default value of "{{query2.data}}". The "LABEL" settings show "Price" as the label and "Price" as the caption. The "INTERACTION" settings show "Disabled" set to "false".

Screenshot 13: Webpage for users to enter their car details when putting it up for sale.

The screenshot shows the final web application for posting car details. It features a dark-themed form with 14 input fields, each labeled with a required indicator (*). The fields correspond to the same categories as in Screenshot 12. A large yellow "Submit" button is positioned at the bottom. The browser's header shows "Car Sale Posting" and "Previewing Latest". The status bar at the bottom indicates "No queries running".

5. Azure Virtual Machine

In order to host this application/website a virtual machine was created in which we cloned a git docker environment which is responsible for hosting our website. This environment is created by retool themselves. A reference to this is mentioned in this [link](#). As mentioned in the documentation we created the virtual machine and added the necessary security and network rules to it so that retool can communicate with Azure. Below screenshots show that we have connected to the virtual machine and check if the docker state is UP. The retool service is hosted on port 3000. To access it we need to put in our public ip address {{public_ip}}/3000.

Screenshot 14, 15 and 16: Azure Virtual Machine setup to host our website.

Resource group (move) : IST615_Project
Status : Running
Location : East US (Zone 1)
Subscription (move) : Visual Studio Enterprise Subscription
Subscription ID : 30959f3c-42af-4cd6-8882-0098c40f879c
Availability zone : 1
Tags (edit) : Click here to add tags

Properties	Monitoring	Capabilities (7)	Recommendations	Tutorials
Virtual machine	Computer name: frontendvm Health state: - Operating system: Linux (ubuntu 20.04) Publisher: canonical Offer: 0001-com-ubuntu-server-focal Plan: 20-04-lts-gen2 VM generation: V2 VM architecture: x64 Agent status: Ready Agent version: 2.8.0.11 Host group: None	Networking Public IP address: 20.115.71.231 Public IP address (IPv6): - Private IP address: 10.1.0.4 Private IP address (IPv6): - Virtual network/subnet: IST615_Project-vnet/default DNS name: Configure		
Size	Size: Standard D2s v3 vCPUs: 2 RAM: 8 GiB			

IP configuration: ipconfig1 (Primary)

Network Interface: frontendvmb29_x1 Effective security rules Troubleshoot VM connection issues Topology
Virtual network/subnet: IST615_Project-vnet/default NIC Public IP: 20.115.71.231 NIC Private IP: 10.1.0.4 Accelerated networking: Enabled

Priority	Name	Port	Protocol	Source	Destination	Action
1000	default-allow-ssh	22	TCP	Any	Any	Allow
1010	AllowAnyHTTPInbound	80	TCP	Any	Any	Allow
1020	AllowAnyHTTPSPInbound	443	TCP	Any	Any	Allow
1030	AllowAnyCustom3000Inbound	3000	Any	Any	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny

```
Last login: Sun Dec  4 21:29:02 on ttys001
(base) mikhailpiotrovskiy@Mikhailis-MBP ~ % ssh -i /Users/mikhailpiotrovskiy/Desktop/frontendvm_key.pem azureuser@20.115.71.231
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-1023-azure x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

System information as of Mon Dec  5 06:10:33 UTC 2022

System Load:          1.38
Usage of /:           39.5% of 28.89GB
Memory usage:        12%
Swap usage:          0%
Processes:            168
Users currently in:   0
IPv4 address for br-3217eefaf892: 172.21.0.1
IPv4 address for br-33ea8c341c55: 172.19.0.1
IPv4 address for br-4e8bb0b167dd: 172.20.0.1
IPv4 address for br-ba5b9cd76e0: 172.17.0.1
IPv4 address for docker0: 172.17.0.1
IPv4 address for eth0:  10.1.0.4

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
just raised the bar for easy, resilient and secure K8s cluster deployment.
https://ubuntu.com/engage/secure-kubernetes-at-the-edge

16 updates can be applied immediately.
8 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

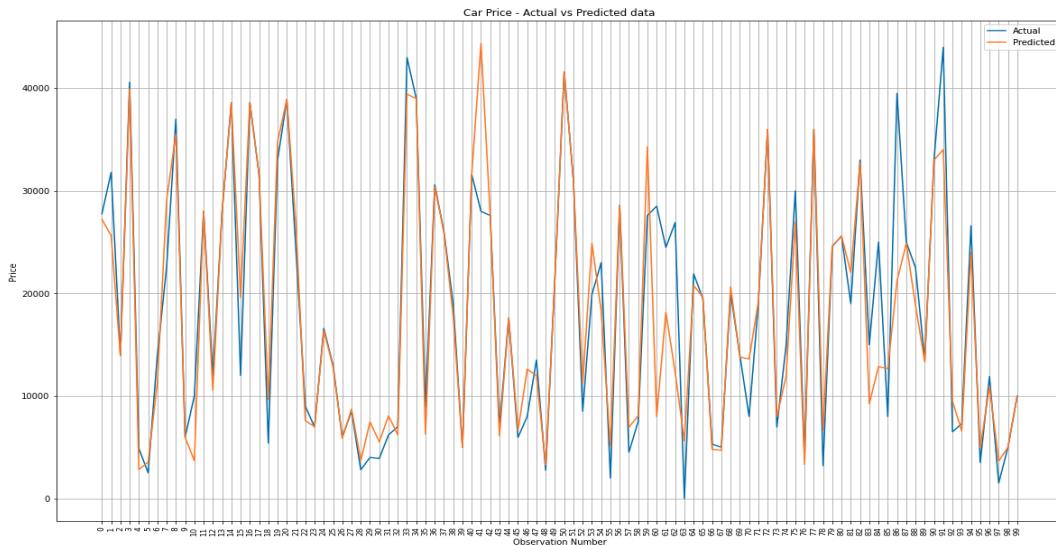
Last login: Mon Dec  5 06:02:14 2022 from 72.230.67.73
[azureuser@frontendvm:~/retool-onpremises]$ cd retool-onpremises
[azureuser@frontendvm:~/retool-onpremises]$ sudo docker-compose ps
          Name      Command     State            Ports
-----  -----
retool-onpremise_api_1    docker-entrypoint.sh bash ...  Exit 0
retool-onpremise_db-connector_1  docker-entrypoint.sh bash ...  Up      3000/tcp, 3001/tcp, 3002/tcp
retool-onpremise_db-ssh-connector_1  docker-entrypoint.sh bash ...  Up      3000/tcp, 3001/tcp, 3002/tcp
retool-onpremise_https-portal_1   /init              ...  Up      0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp
retool-onpremise_jobs-runner_1   docker-entrypoint.sh bash ...  Exit 137
retool-onpremise_postgres_1     docker-entrypoint.sh postgres ...  Exit 0
retool-onpremise_user-postgres_1  docker-entrypoint.sh postgres ...  Exit 0
retool-onpremise_db-ssh-connector_1  docker-entrypoint.sh bash ...  Up      to date
retool-onpremise_db-ssh-connector_1  in up-to-date
Starting retool-onpremise_api_1 ...
Starting retool-onpremise_postgres_1 ...
Starting retool-onpremise_user-postgres_1 ...
Starting retool-onpremise_jobs-runner_1 ...
Starting retool-onpremise_db-ssh-connector_1 ...
retool-onpremise_https-portal_1 is up-to-date
[azureuser@frontendvm:~/retool-onpremises]$ sudo docker-compose up -d
          Name      Command     State            Ports
-----  -----
retool-onpremise_api_1    docker-entrypoint.sh bash ...  Up      0.0.0.0:3000->3000/tcp, :::3000->3000/tcp, 3001/tcp, 3002/tcp
retool-onpremise_db-connector_1  docker-entrypoint.sh bash ...  Up      3000/tcp, 3001/tcp, 3002/tcp
retool-onpremise_db-ssh-connector_1  docker-entrypoint.sh bash ...  Up      3000/tcp, 3001/tcp, 3002/tcp
```

Once this is done we can go ahead and see the hosted website on the public_ip address and port 3000. The website can also be managed using the same VM.

6. Azure Databricks

This would be used for the purpose of python scripting, analysis and model creation in our project. We wrote a script to integrate Azure Blob storage and Azure Databricks. Using this script, we can read data from our container in blob storage to do further analysis on our dataset. Some additional data cleansing was done and new columns were added to add on to the features to build models .Three models - Linear Regression, Random Forest and XGBoost were built to predict estimated prices for users' cars that they need to post online for sale. Our best predicting model was Random Forest and it is being used to predict prices for these used cars. Predicted prices data is stored in csv format in another blob storage named 'predictions.'

Screenshot 17: Graph for comparison of Random Forest Model price predictions and actual prices of cars.



Tasks Completed:

We were successfully able to implement the integration between Azure Blob Storage and SQL Database and create and host a website on a Virtual Machine. Finally we were also able to do some data analysis for our data and predict prices of cars using Azure Databricks.

Issues encountered:

- The biggest challenge that we faced was creating a front-end for our website as it is not in our skillset to do it. The initial plan to do it using App Services (Web App + Database) integration was dropped as it would take deep knowledge of .NET and some HTML and JavaScript. In order to tackle this issue we used an external service Retool <https://retool.com/use-case/sql-front-end> which was finally hosted on an Azure Virtual Machine.
- The second challenge that we faced was connecting Azure Databricks to Power BI to showcase our findings from our data and display our predicted prices. Due to unavailability of proper power bi licensing, we were unable to go around this issue or come up with a solution in time.
- Managing the cost for each service was extremely difficult due to the dataset being large and we had a lot of computes to handle.

Future Scope:

- After proper testing of our model built, include our predicted prices in the price estimator webpage of our website.
- Establish connection to PowerBI and build dashboards to display our data analysis, predicted prices and our model performance over time.