MIE1628HY Cloud-based Data Analytics

Assignment 5

Faculty of Engineering – University of Toronto – Graduate Studies

Spring/Summer 2023

Submitted by: Zorina Song

UTORid: songzir1

Student#: 1005637198

Email: zo.song@mail.utoronto.ca

Submission date: 08/13/2023

Team member: Jingcheng Zhao,

UTORid: zhaoj116

Student#: 1003797965

Email: Jensen.zhao@mail.utoronto.ca

Part A:

1. [Marks: 5] Explain below the 5 components shown in orange boxes. Explain which Azure components you will use where in this big data architecture and why.

Explanation of the 5 components:

- Azure Data Lake: cloud platform, which allow us to store and analyze large volumes of data in various formats, such as structured, semi-structured, and unstructured data. It provides capabilities for big data processing, analytics, and machine learning.
- Azure Databricks: fully managed first-party service which could enables an open data lakehouse in Azure. Offers an integrated environment to enable multiple departments work together on big data and machine learning projects. It provides tools for data preprocessing, analytics, and machine learning model development, leveraging the power of distributed computing and parallel processing.
- Azure Data Factory: a cloud ETL service which can be used to scale-out serverless data integration. It can also support data movement across on-premises and cloud environments and facilitates data transformations.
- Azure Synapse Analytics: a cloud-based analytics service that combines big data and data warehousing capabilities into a single unified platform. It allows us to analyze large volumes of data using both on-demand and provisioned resources. Synapse Analytics integrates data storage, data processing, and data integration, enabling us to perform complex analytics tasks on massive datasets.
- Azure Cosmos DB: a globally distributed, multi-model database service, which can handle massive amounts of data with low-latency and high

availability for managing data at large scales. Various data models, including document, key-value, graph, and column-family, can be supported, allowing us to choose the right model for our application.

Which Azure components you will use in this big data architecture:

- Ingest Data: Azure Data Factory

Ideal for ingesting data from various sources into the data processing pipeline, as it provides connectors to a wide range of data sources and destinations and allows us to define data pipelines, schedule data movement, and manage data workflows effectively.

- Data Store: Azure Data Lake Storage

It is optimized for storing and analyzing large amounts of data in its raw, unprocessed form. We can accommodate both structured and unstructured data, making it suitable for big data scenarios.

- Prepare and transform data: Azure Databricks

Due to its distributed processing capabilities and interactive notebooks, and it can handle complex data processing tasks at scale, making it well-suited for cleaning, transforming, and enriching large datasets before analysis.

- Model and serve data: Azure Synapse Analytics

As a fully managed analytics service that brings together big data and data warehousing into a single unified platform. It's designed for querying and analyzing large datasets at scale. Allows us to build data warehouses and analytical pipelines that integrate data from various sources.

2. [Marks: 5] Explain how Stream Analytics works in Azure.

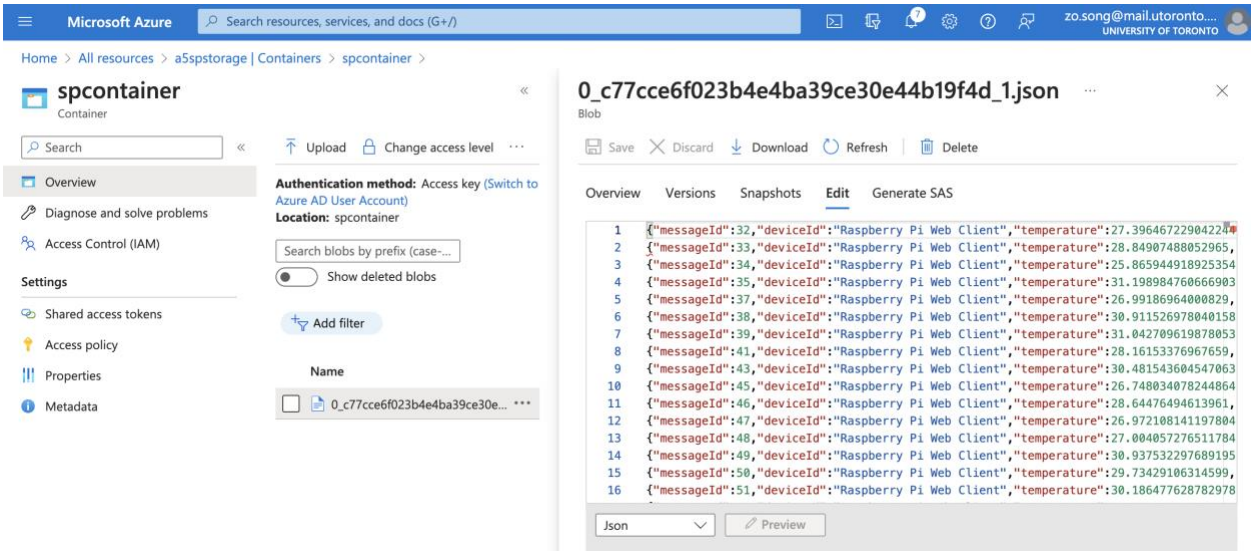
Ans:

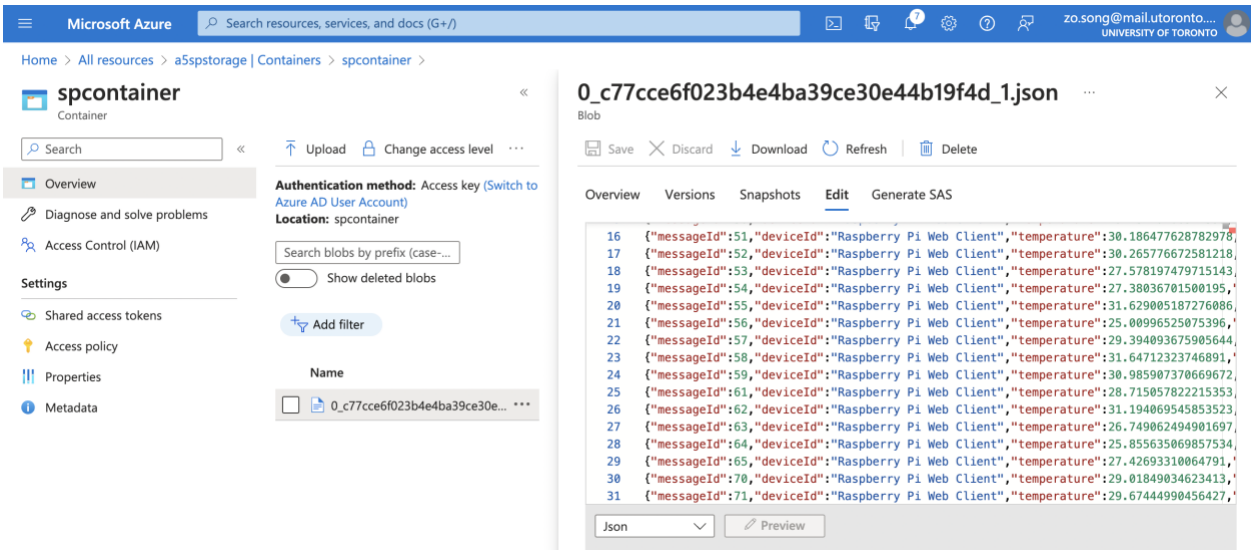
As a real-time data processing service in Azure, it takes in streaming data from various sources, applies SQL-like queries for transformations, aggregations, and filtering, and then outputs the results to different destinations. We can use it for analyzing live data, providing real-time insights and integration with other Azure services.

Data Ingestion ==>processing ==> query execution
==> windowing and time Management (allowing us to perform operations on data over specific time intervals)
==> output sinks (destinations where the processed data is stored or sent for further analysis)
==> scaling and monitoring

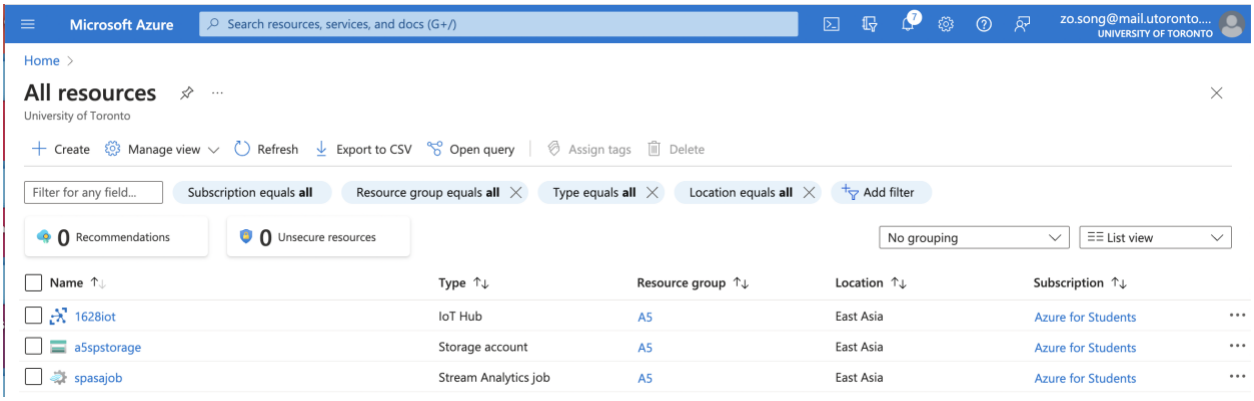
3. [Marks: 10] Deploy all the resources in Azure Portal. Implement a Stream Analytics job by using the Azure portal.

ANS:





Deployed Resources:



Part B:

Question 1:

The dataset contains the following features for each concrete mixture:

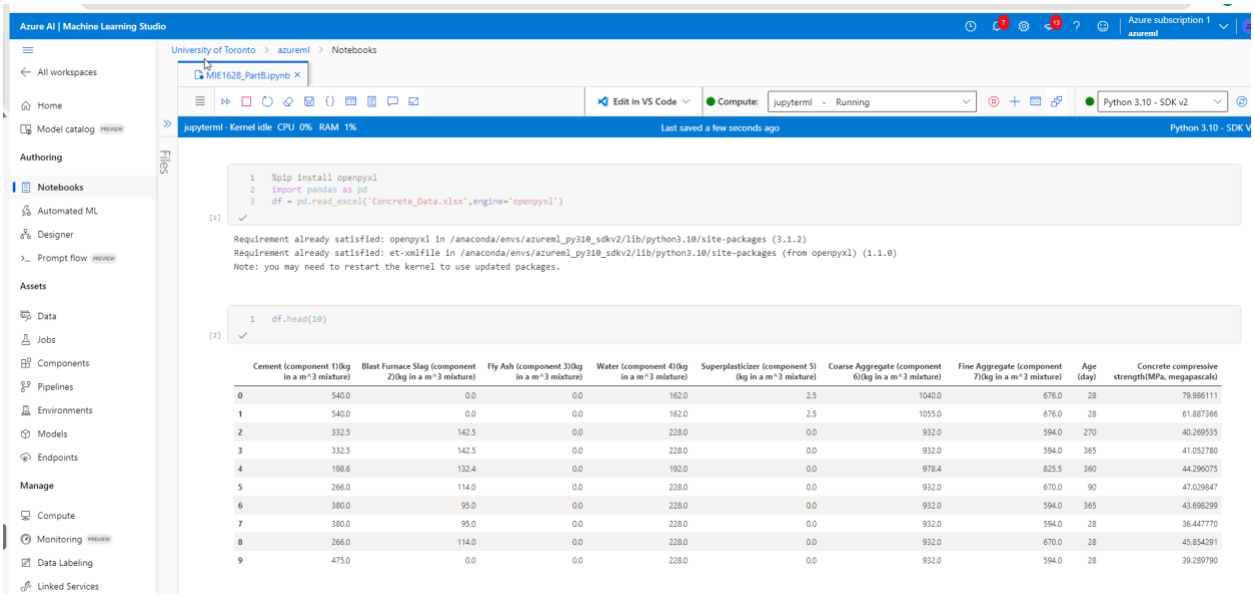
1. Cement (component 1): The amount of cement(kg) in the mixture(m3).
2. Blast Furnace Slag (component 2): The amount of blast furnace slag(kg) in the mixture(m3).
3. Fly Ash (component 3): The amount of fly ash(kg) in the mixture(m3).
4. Water (component 4): The amount of water(kg) in the mixture(m3).
5. Superplasticizer (component 5): The amount of superplasticizer(kg) in the mixture(m3).
6. Coarse Aggregate (component 6): The amount of coarse aggregate(kg) in the mixture(m3).
7. Fine Aggregate (component 7): The amount of fine aggregate(kg) in the mixture(m3).
8. Age: The age of the concrete in days.

Problem Statement:

Given the dataset of concrete mixtures with their corresponding components and their actual compressive strength (measured in MPa), our task is to build a regression model capable of accurately predicting the concrete's compressive strength for any given mixture of components and its age. The successful development of such a model can be valuable in construction and civil engineering, as it would help engineers, contractors, and researchers estimate the strength of concrete structures without conducting time-consuming and costly physical tests. The model can aid in optimizing concrete mixtures for specific applications, ensuring safety, durability, and cost-effectiveness in construction projects.

Question 2:

Explore Dataset:



There is a description of the features in the above section.

Through our exploration, we have identified 8 input features and 1 target feature in our dataset. These features are all in numerical and continuous formats. The only input feature in integer is the 'age' column, which represents the number of. There are a total of 1030 entries in our sample dataset.

Upon analyzing the initial 10 rows of data, we observed instances where certain columns contained values of 0. These columns include measurements like the amount of blast furnace slag (in kg/m^3) and fly ash (in kg/m^3).

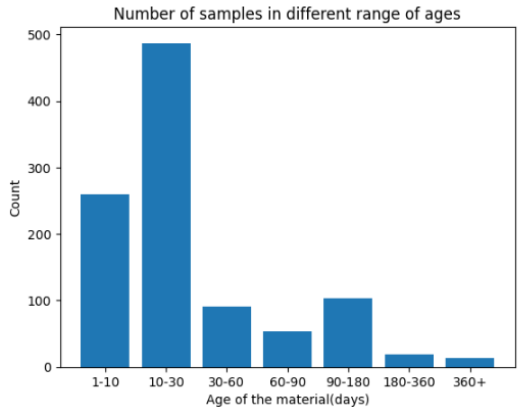
Furthermore, we have computed fundamental statistics such as the mean, standard deviation, minimum, and maximum values for each feature. These statistics provide valuable insights into the data distribution, aiding in our understanding of the dataset and guiding us in our exploration for subsequent steps.

Here are the 6 plots with explanations:

Plot1: Number of samples in different range of ages

```
[6]
import matplotlib.pyplot as plt
bins = [1, 10, 30, 60, 90, 180, 360, float('inf')]
labels = ['1-10', '10-30', '30-60', '60-90', '90-180', '180-360', '360+']
df_new['Age_Count'] = pd.cut(df_new['Age'], bins=bins, labels=labels)
# Count the occurrences of each range
range_counts = df_new['Age_Count'].value_counts().sort_index()

# Plotting
plt.bar(range_counts.index, range_counts.values)
plt.xlabel('Age of the material(days)')
plt.ylabel('Count')
plt.title('Number of samples in different range of ages')
plt.show()
```

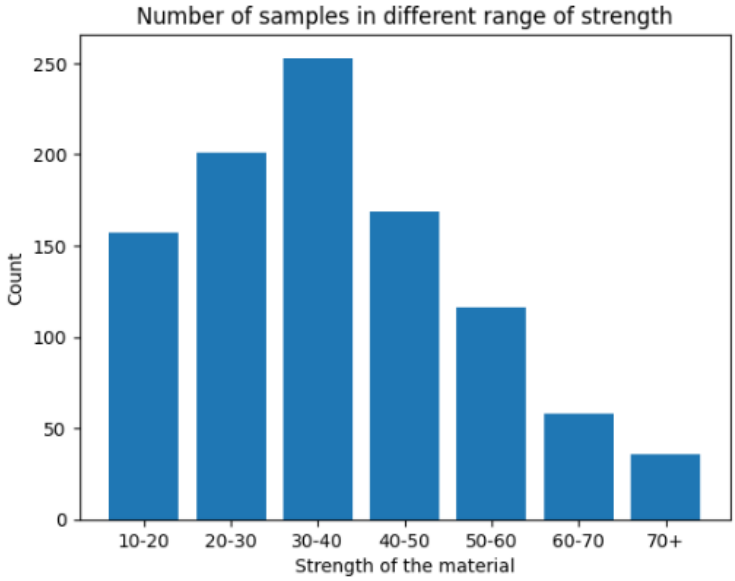


Explain:

In summary, the graph shows that most samples are concentrated in the 1 to 30-day age range, with the peak count in the 10-30 days range which is almost 500 counts and there are over 250 counts in the lower 1 to 10-day age range. The count generally diminishes with greater age ranges beyond 30 days, but the "90-180 days" range presents itself as the third most frequent range, with around 100 counts. This prominence might be influenced by the wider span of this interval compared to the previous ones, potentially encompassing a broader range of samples.

Plot2: Number of samples in different range of strength

```
[7]
1   bins = [10, 20, 30, 40, 50, 60, 70, float('inf')]
2   labels = ['10-20', '20-30', '30-40', '40-50', '50-60', '60-70', '70+']
3   df_new['Strength_Count'] = pd.cut(df_new['Concrete_Strength'], bins=bins, labels=labels, right=False)
4   # Count the occurrences of each range
5   range_counts = df_new['Strength_Count'].value_counts().sort_index()
6
7   # Plotting
8   plt.bar(range_counts.index, range_counts.values)
9   plt.xlabel('Strength of the material')
10  plt.ylabel('Count')
11  plt.title('Number of samples in different range of strength')
12  plt.show()
```



Explain:

The plot shows a slight right-skewed distribution. There are three main phases with increased strength:

1. Increasing strength: As the strength value increases from 10 MPa, the number of samples also rises. This indicates that, within the strength 10-40 MPa, higher strength values are more common among the samples.
2. Peak frequency: The most frequent strength values are found in the range of 30-40 MPa. Around 250 samples have strength within this range, which makes it the peak of the distribution.
3. Decreasing trend: Following the 30-40 MPa range, the counts of samples start to decrease. This means that fewer samples have strength values beyond this point in our samples.

Plot3: Ratio of material use fly ash and superplasticizer

```
ash_nonzero_sup_zero = ((df_new['Fly_Ash'] != 0) & (df_new['Superplasticizer'] == 0)).sum()
sup_nonzero_ash_zero = ((df_new['Superplasticizer'] != 0) & (df_new['Fly_Ash'] == 0)).sum()
both_zero = ((df_new['Fly_Ash'] == 0) & (df_new['Superplasticizer'] == 0)).sum()
both_nonzero = ((df_new['Fly_Ash'] != 0) & (df_new['Superplasticizer'] != 0)).sum()
```

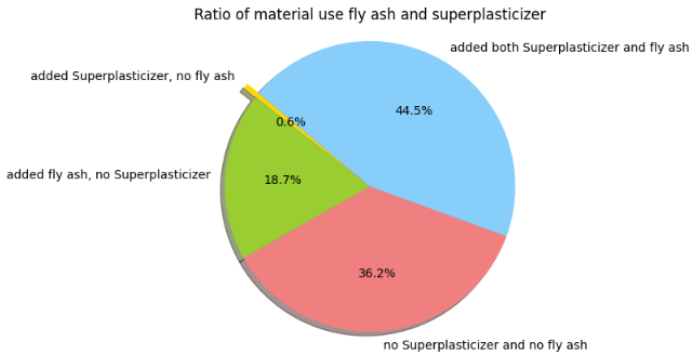
[9] ✓

```
# Create a pie chart
labels = ['added Superplasticizer, no fly ash', 'added fly ash, no Superplasticizer', 'no Superplasticizer and no fly ash', 'added both Superplasticizer and fly ash']
sizes = [ash_nonzero_sup_zero, sup_nonzero_ash_zero, both_zero, both_nonzero]
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']
explode = (0.1, 0, 0, 0)  # explode 1st slice

plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.title('Ratio of material use fly ash and superplasticizer')
plt.show()
```

[10] ✓



Explain:

The most common material usage scenario involves adding both fly ash and superplasticizer components to the mixture, constituting approximately 44.5% of the samples.

Interestingly, the second-largest group of samples consists of those that do not include either fly ash or superplasticizer, which accounts for a substantial portion of 36.2%.

Around 18.7% of the samples added fly ash only, without the addition of superplasticizer, while only a light fraction, about 0.6%, of the samples use superplasticizer as the exclusive component in the mixture.

Plot4. Relationship between the amount of cement and concrete strength

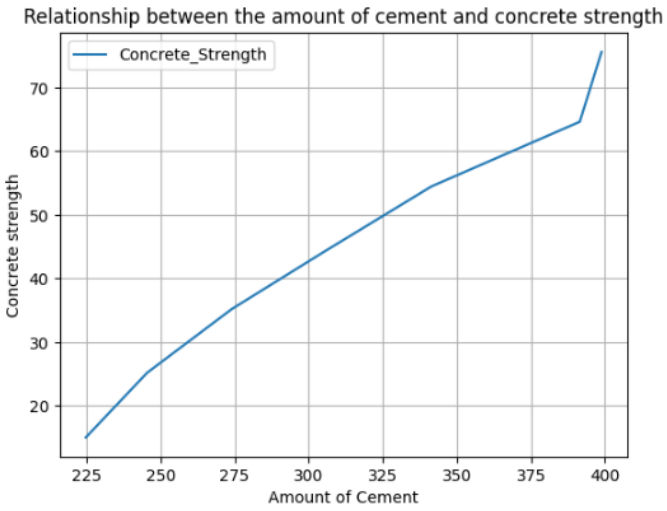
```
df_s = df_new.drop('Age_Count',axis = 1).groupby("Strength_Count").mean()
df_s = df_s.reset_index(drop=False)
```

11]

```
df_s.plot(x='Cement', y='Concrete_Strength', kind='line')

plt.ylabel('Concrete strength')
plt.xlabel('Amount of Cement')
plt.title('Relationship between the amount of cement and concrete strength')
plt.grid(True)
plt.show()
```

12]



Explain:

Most data points form a pattern that ascends as the amount of cement increases, which indicates a positive linear relationship between the amount of cement(kg/m^3) and the concrete strength (Mpa). In other words, as the amount of cement used in the mixture increases, the concrete strength tends to increase as well. The pattern observed aligns with our existing knowledge that an increased amount of cement is likely to result in higher concrete strength.

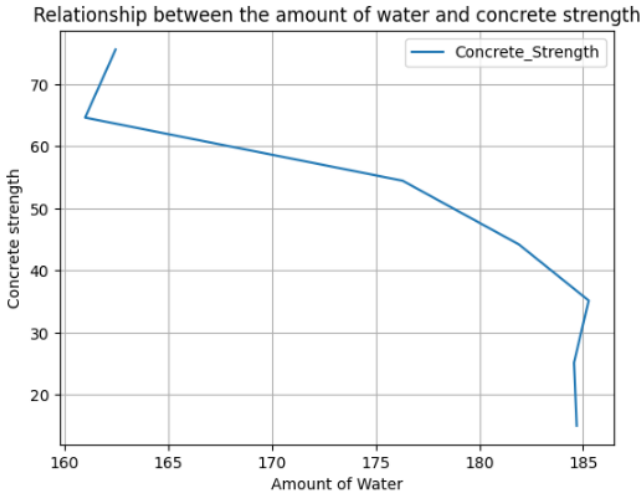
However, the plot shows a turning point around 380 kg/m^3 of cement. Beyond this point, the rate of strength increases experiences a sudden and dramatic rise. This suggests that the rate of strength improvement becomes notably higher for cement quantities exceeding 380 kg/m^3.

Plot5. Relationship between the amount of water and concrete strength

+ Code + Markdown

```
1  df_s.plot(x='Water', y='Concrete_Strength', kind='line')
2
3  plt.ylabel('Concrete strength')
4  plt.xlabel('Amount of Water')
5  plt.title('Relationship between the amount of water and concrete strength')
6  plt.grid(True)
7  plt.show()
```

[13] ✓

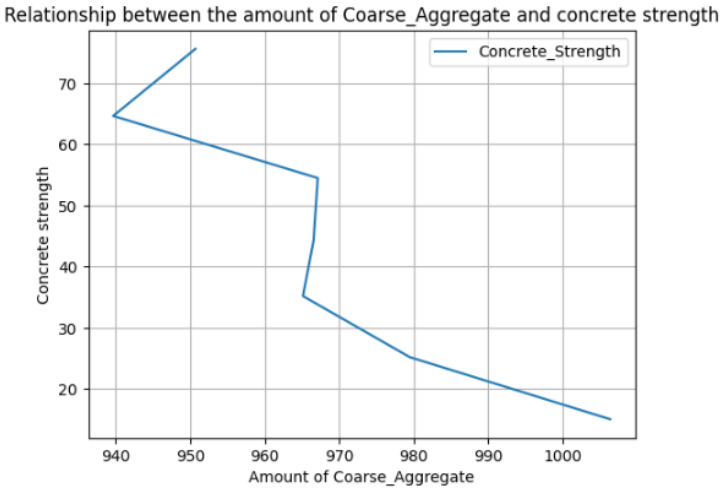


Explain:

In general, there is a negative linear relationship between the amount of water and concrete strength. As the amount of water increases, the concrete strength tends to decrease. Some data points at the extreme ends of the water amount spectrum do not conform to the general trend. These outliers or noises might represent unusual or non-standard scenarios that deviate from the typical relationship. Around 176 kg/m^3 of water, a turning point was observed that beyond this point, higher amounts of water seem to lead to a more rapid decrease in concrete strength.

Plot6. Relationship between the amount of coarse aggregate and concrete strength

```
[44]
1  df_s.plot(x='Coarse_Aggregate', y='Concrete_Strength', kind='line')
2
3  plt.ylabel('Concrete strength')
4  plt.xlabel('Amount of Coarse_Aggregate')
5  plt.title('Relationship between the amount of Coarse_Aggregate and concrete strength')
6  plt.grid(True)
7  plt.show()
```



Explain:

Overall, there seems to be a negative relationship between the amount of coarse aggregate and concrete strength, such that the concrete strength tends to decrease as the amount of coarse aggregate increases. Within the interval of 940 to 950 kg/m^3 of coarse aggregate, there are fluctuations in concrete strength, suggesting there are variations in strength within this specific range of coarse aggregate amounts. Besides, a dramatically decreases in the slope of the line occurs between 960 and 970 kg/m^3 of coarse aggregate, indicating that changes in coarse aggregate beyond this range might have a more pronounced impact on concrete strength. Beyond the drop point, the trend indicates a more steady and consistent decrease in concrete strength as coarse aggregate amounts increase.

Question 3:

3.1 Replacing NaN Values with Column Averages:

Question3

Change the columns name to more readable form

```
[23]
1  df_predict = pd.read_excel('Concrete_Data.xlsx',engine='openpyxl')
2
3  new_column_names = ['Cement', 'Blast_Furnace_Slag', 'Fly_Ash', "Water", "Superplasticizer", "Coarse_Aggregate", "Fine_Aggregate", "Age", "Concrete_Strength"]
4  df_predict.columns = new_column_names
```

Fill all NAN values with average values

```
[24]
1  # Replace NaN values with the average of their respective columns
2  df_predict.fillna(df_predict.mean(), inplace=True)
```

NaN values (missing values) can disrupt the training of the model in the further step. Replacing them with the average of their respective columns is a common strategy to ensure that the data maintains its general distribution while filling in the gaps.

3.2 Removing Duplicates:

Removing Duplicates

```
[25]
1  # Remove duplicates
2  df_predict = df_predict.drop_duplicates()
```

Removing duplicates ensures that each data point is unique and representative, and result in an overly optimistic view of the model's performance.

3.3 **Standardizing the Dataset:**

```
from sklearn.preprocessing import StandardScaler

# Standardize the dataset
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_predict)

# Convert the scaled data array back to a DataFrame
scaled_df = pd.DataFrame(scaled_data, columns=df_predict.columns)

print("Original DataFrame:")
print(df_predict.head(5))
print("\nStandardized DataFrame:")
print(scaled_df.head(5))
```

[26] ✓

```
Original DataFrame:
   Cement  Blast_Furnace_Slag  Fly_Ash  Water  Superplasticizer  \
0   540.0                 0.0      0.0  162.0               2.5
1   540.0                 0.0      0.0  162.0               2.5
2   332.5               142.5      0.0  228.0               0.0
3   332.5               142.5      0.0  228.0               0.0
4   198.6               132.4      0.0  192.0               0.0

   Coarse_Aggregate  Fine_Aggregate  Age  Concrete_Strength
0            1040.0           676.0   28          79.986111
1            1055.0           676.0   28          61.887366
2             932.0           594.0  270          40.269535
3             932.0           594.0  365          41.052780
4             978.4           825.5  360          44.296075
```

Standardization scales the features to have a mean of 0 and a standard deviation of 1. As the differences in the amounts of components (measured in kg/m^3) across the samples are noticeable, this step ensures that each feature contributes to the model's learning process equally and prevents features with larger numerical values from dominating features with smaller values.

After standardizing the data, working with it in a DataFrame format is a common data structure used in Pandas, as it’s often more convenient to work with DataFrames in further analysis or use in models.

3.4 **Train-Test Split:**

Train-Test Split

```
[31]
from sklearn.model_selection import train_test_split

y = scaled_df['Concrete_Strength']
X = scaled_df.drop(columns = 'Concrete_Strength')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train:")
print(X_train)
print("\nX_test:")
print(X_test)
print("\ny_train:")
print(y_train)
print("\ny_test:")
print(y_test)
```

```
X_train:
       Cement  Blast_Furnace_Slag   Fly_Ash     Water  Superplasticizer  \
78   2.422701           -0.836469 -0.865363 -1.888146          3.746798
29   1.882876           -0.836469 -0.865363  2.153088         -1.019442
280 -0.015141           -0.836469  0.652195 -1.006763          0.980012
507  1.508930           -0.836469 -0.865363  1.684267         -1.019442
652 -1.072258            2.068521 -0.865363  1.004477         -1.019442
..        ...                 ...       ...       ...               ...
106  0.568311            1.357948 -0.865363 -0.336350          0.586206
270 -1.052314           -0.346267  1.070735 -1.113185          0.810997
860 -1.204577            0.847078  0.895434 -0.191015          0.332682
435 -1.089517            0.655502  1.193055 -0.328849          0.346204
102  0.073552            1.496115 -0.865363 -1.752188          0.873532

     Coarse Aggregate  Fine Aggregate       Age
```

A train-test split allows us to train the model on one subset of the data (training set) and test it on another subset (testing set) that it has not seen during training. It is essential to evaluate the model's performance on unseen data to understand its generalization ability. Here we choose 80% as training set and the remaining 20% as testing set, which is a common choice.

Question 4:

Machine learning model1: Decision Tree Regressor

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
# Train Decision Tree Regressor model
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)


# Make predictions
dt_pred = dt_model.predict(X_test)

# Calculate RMSE
dt_rmse = mean_squared_error(y_test, dt_pred, squared=False)

# Calculate MSE
dt_mse = mean_squared_error(y_test, dt_pred)

# Calculate MAE
dt_mae = mean_absolute_error(y_test, dt_pred)

# Display MSE and MAE values
print(f"Decision Tree MSE: {dt_mse:.2f}")
print(f"\nDecision Tree RMSE: {dt_rmse:.2f}")
print(f"\nDecision Tree MAE: {dt_mae:.2f}")
```

```
Decision Tree MSE: 0.15

Decision Tree RMSE: 0.39

Decision Tree MAE: 0.24
```

Machine learning model2: Random Forest Regressor

```
# Train Random Forest Regressor model
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)

# Make predictions
rf_pred = rf_model.predict(X_test)

# Calculate RMSE
rf_rmse = mean_squared_error(y_test, rf_pred, squared=False)

# Calculate MSE
rf_mse = mean_squared_error(y_test, rf_pred)

# Calculate MAE
rf_mae = mean_absolute_error(y_test, rf_pred)


# Display MSE, RMSE and MAE values
print(f"Random Forest MSE: {rf_mse:.2f}")
print(f"\nRandom Forest RMSE: {rf_rmse:.2f}")
print(f"\nRandom Forest MAE: {rf_mae:.2f}")
```

```
Random Forest MSE: 0.10

Random Forest RMSE: 0.32

Random Forest MAE: 0.21
```

The reasons why these 2 algorithms are chosen:

We are facing a regression problem; therefore, the selection of the algorithms is limited to those can be used on continuous data type. Among the available machine learning algorithms. I selected decision tree regressor and random forest regressor.

The reasons why the decision tree regression is selected:

1. **Interpretability**: Decision trees provide a clear visualization of how decisions are made based on component values. This helps engineers and researchers understand the relationships between individual components and concrete strength.
2. **Component Interaction**: Decision trees can capture complex interactions between components. They allow for the identification of specific component thresholds that lead to changes in material strength.
3. **Feature Importance**: Engineers can easily identify the most critical components that influence material strength, aiding in material design and optimization.

The reasons why the random forest regression is selected:

1. **Improved Accuracy**: Random forests combine predictions from multiple decision trees, leading to better prediction accuracy by capturing various interactions in the data.
2. **Robustness**: Material strength prediction might involve noisy data or outliers. Random forests handle such variations well due to the ensemble averaging of multiple trees.
3. **Generalization**: Random forests generalize better to new data by reducing overfitting that can occur with a single decision tree.

The comparison between decision tree regression and random forest regression:

We evaluated Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) for both Decision Tree Regression and Random Forest Regression models. These metrics are commonly used to evaluate the performance of regression models.

1. **Mean Squared Error (MSE)**:
    - MSE measures the average squared difference between predicted and actual values. A lower MSE indicates a better fit to the data.
    - The Random Forest model has a lower MSE (0.1) compared to the Decision Tree model (0.15). This suggests that the Random Forest's ensemble approach has produced more accurate predictions on average.
2. **Root Mean Squared Error (RMSE)**:
    - RMSE is the square root of MSE and represents the average prediction error in the same units as the target variable.
    - The Random Forest model has a lower RMSE (0.32) than the Decision Tree model (0.39). This again indicates that the Random Forest model's predictions are closer to the actual values.

3. **Mean Absolute Error (MAE)**:
    - MAE measures the average absolute difference between predicted and actual values.
    - The Random Forest model has a slightly lower MAE (0.21) compared to the Decision Tree model (0.24), which means that the Random Forest predictions have a smaller absolute error on average.

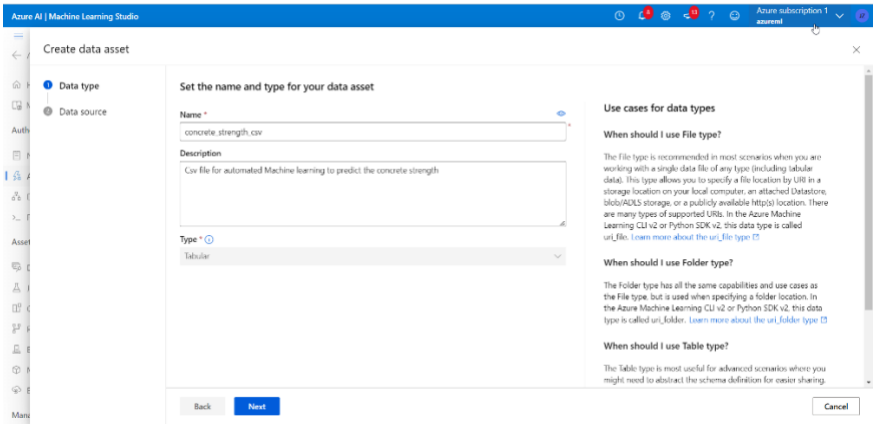
In summary, the results suggest that the Random Forest Regression model is better than the Decision Tree Regression model in terms of prediction accuracy. The lower values of MSE, RMSE, and MAE for the Random Forest model indicate that it provides more accurate predictions on average. This improvement can be attributed to the ensemble nature of the Random Forest, which combines multiple decision trees to achieve better generalization and mitigate overfitting.

Question 5.

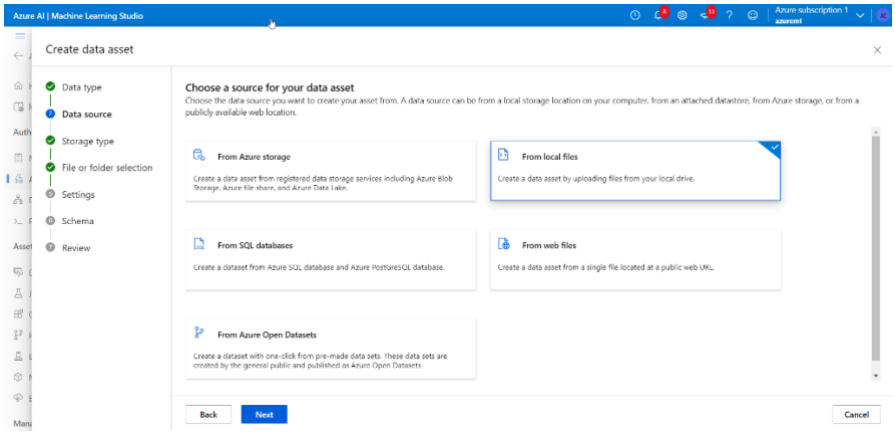
We choose to apply Automated ML on Microsoft Azure for our data set, here are the steps and results:

Setup procedure:

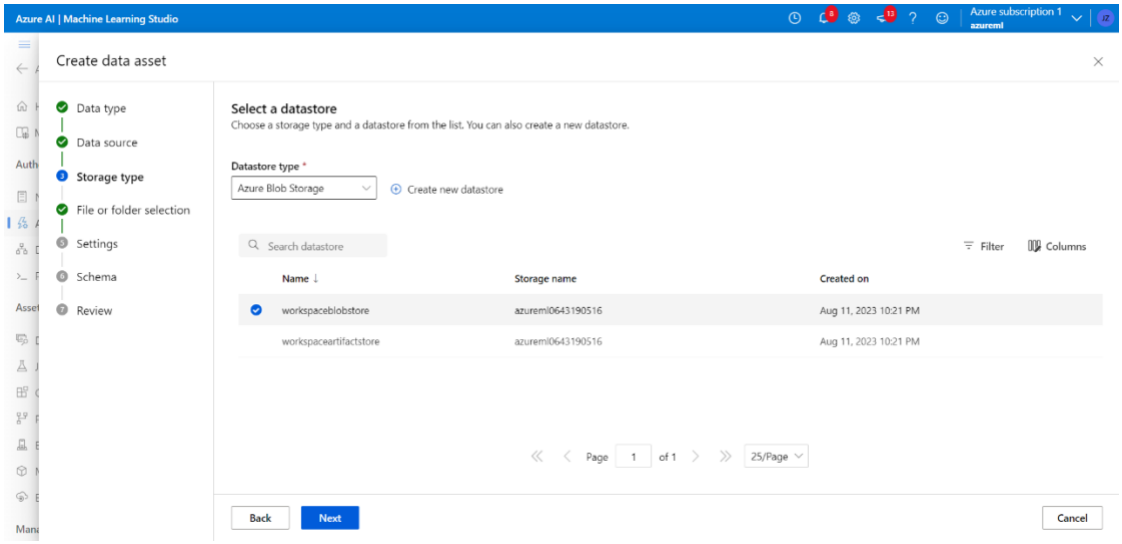
Step1: Create Data Asset (Data Type)



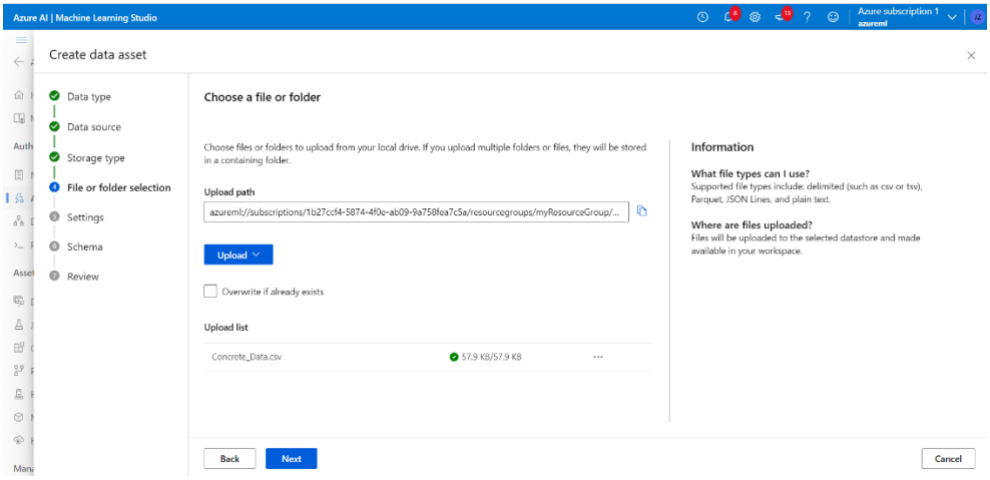
Step2: Create Data Asset (Data Source)



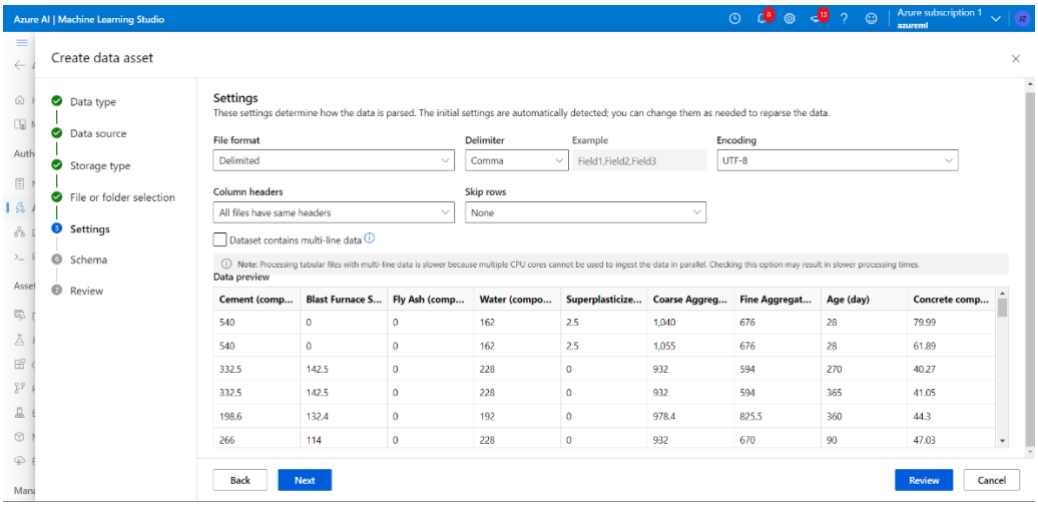
Step3: Create Data Asset (Storage type)



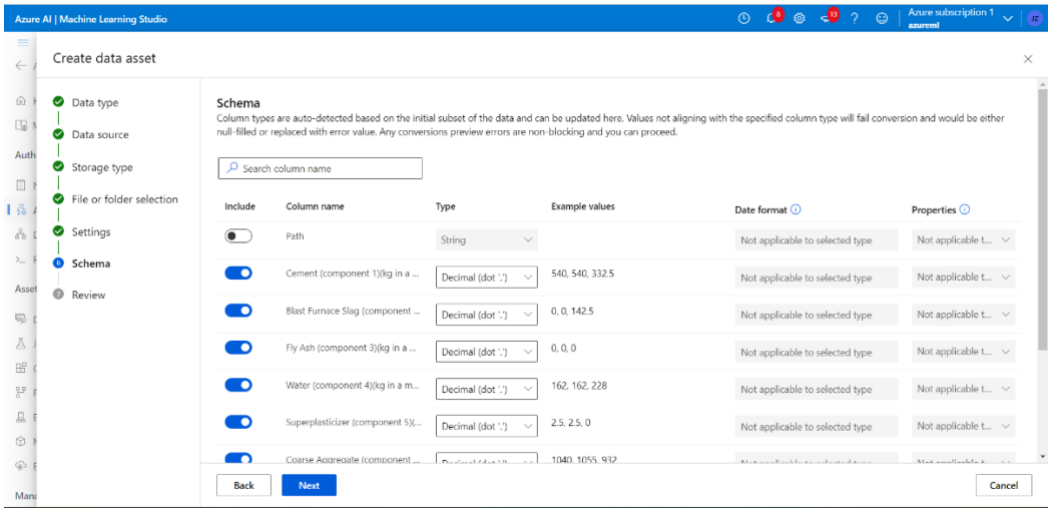
Step4: Create Data Asset ()



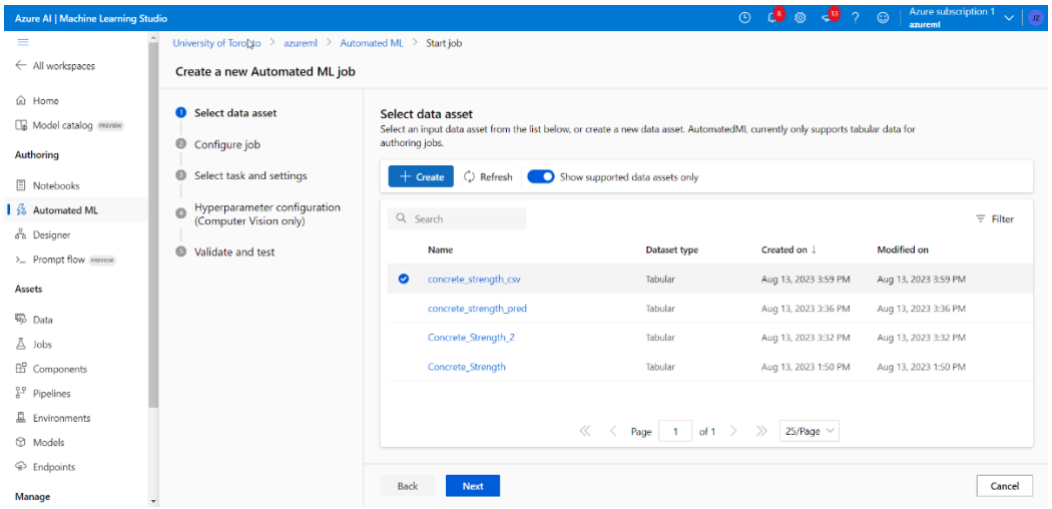
Step5: Create Data Asset (Setting)



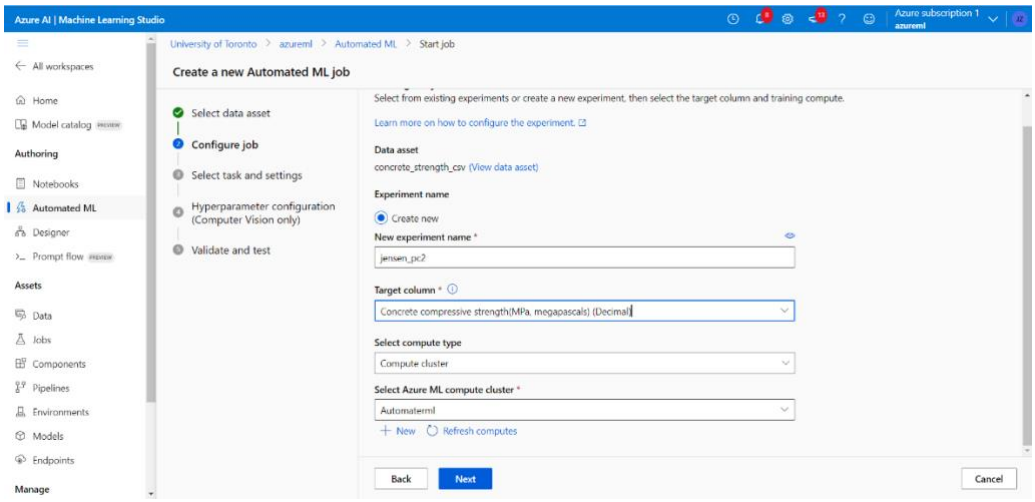
Step6: Create Data Asset (Schema)



Step7: Select Data Asset

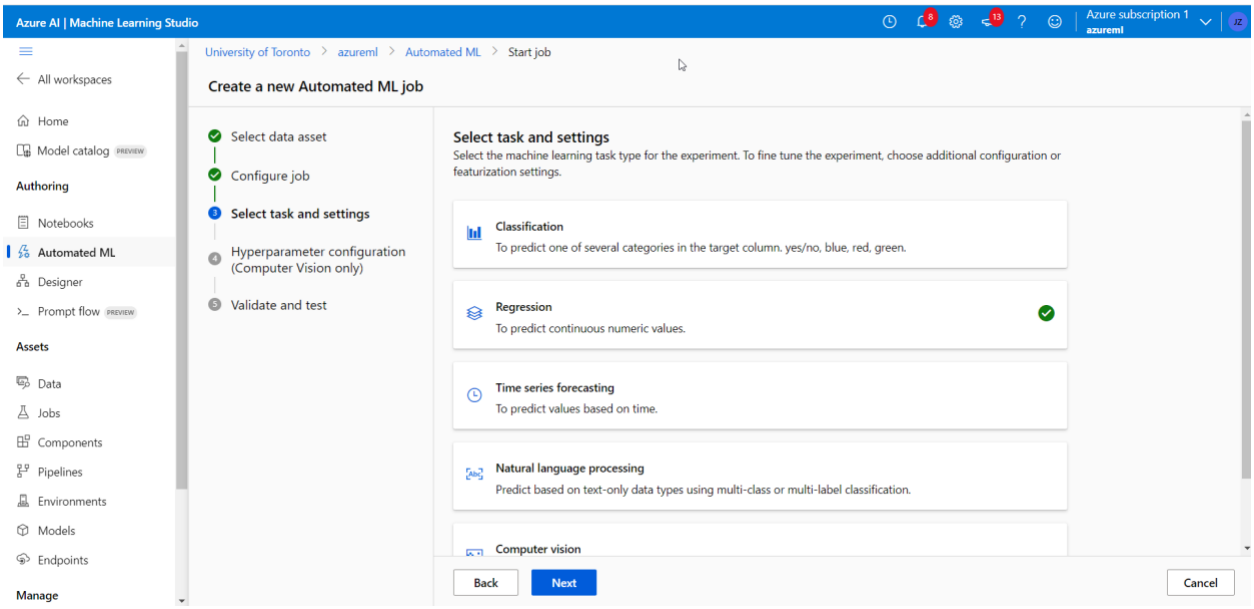


Step8: Configure job



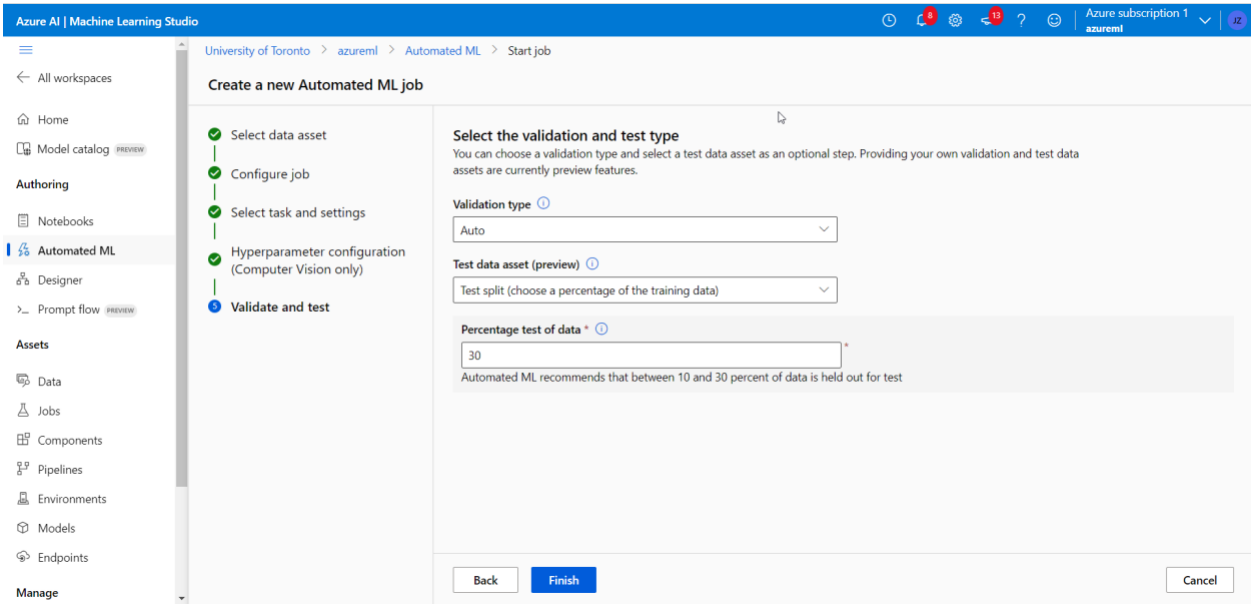
Step9: Select task and settings.

I selected regression.

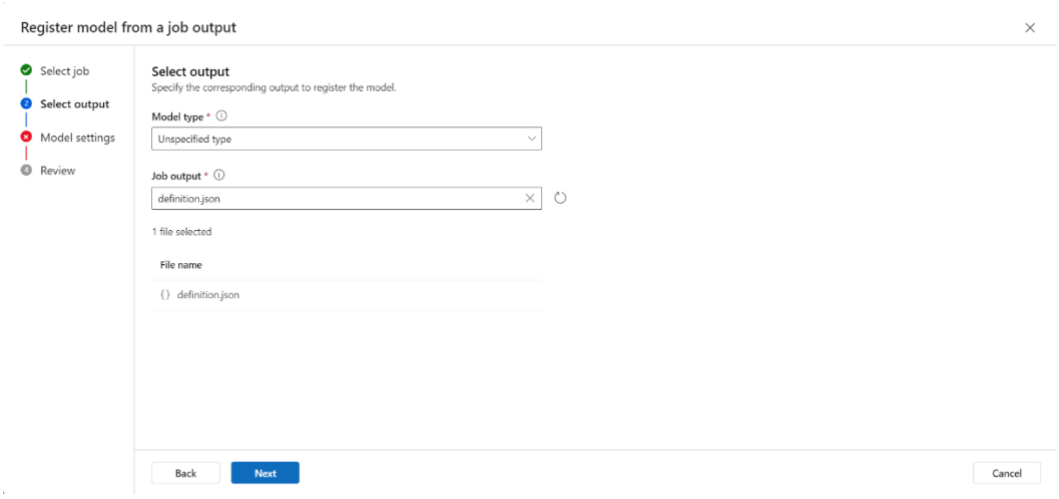


Step10: Hyperparameter configuration

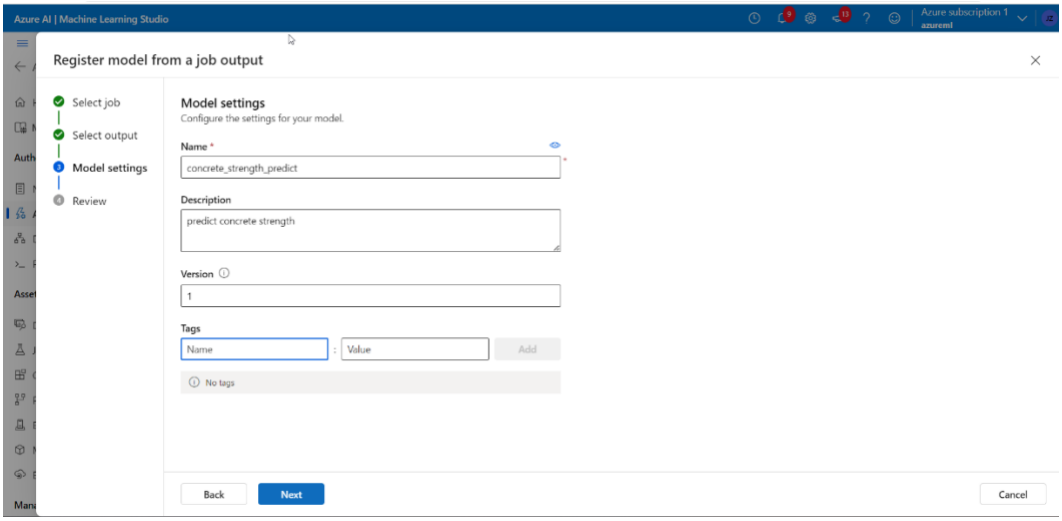
The train test split ration is set to 70:30.



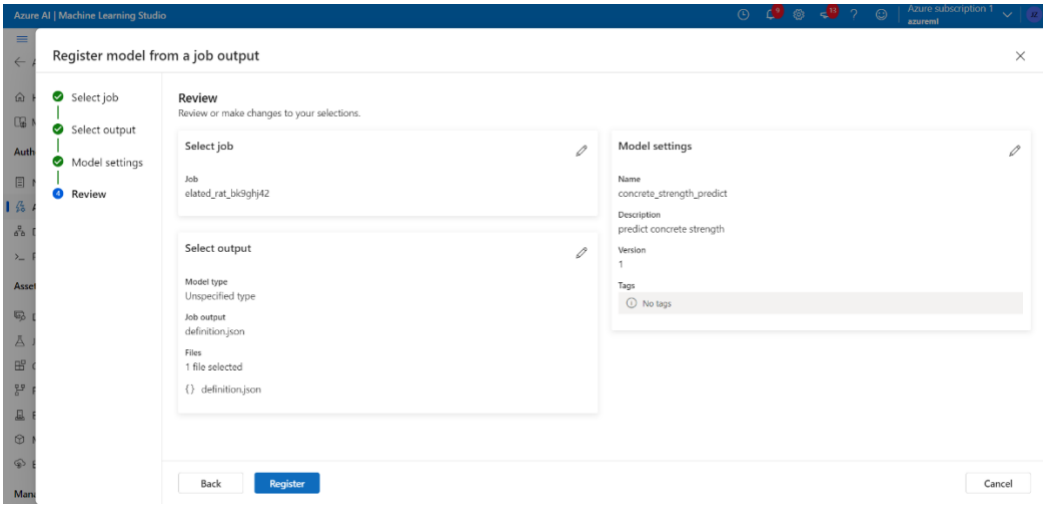
Step11: Register Model (Select output)



Step12: Register Model (Model Settings)

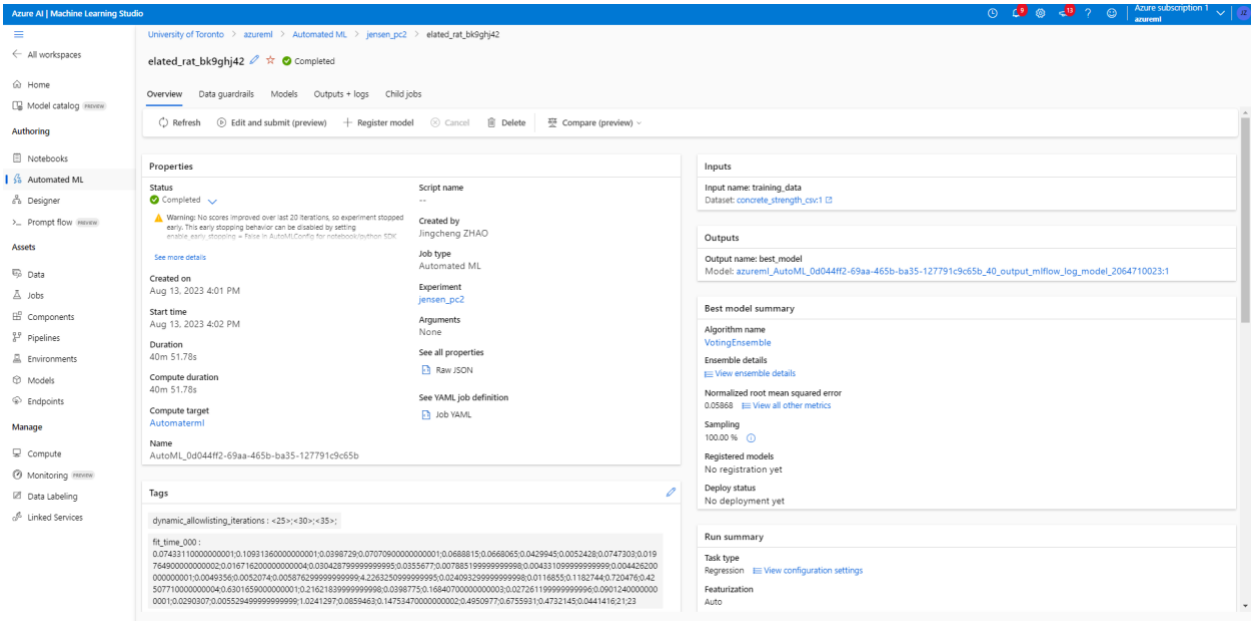


Step13: Register Model (Review)

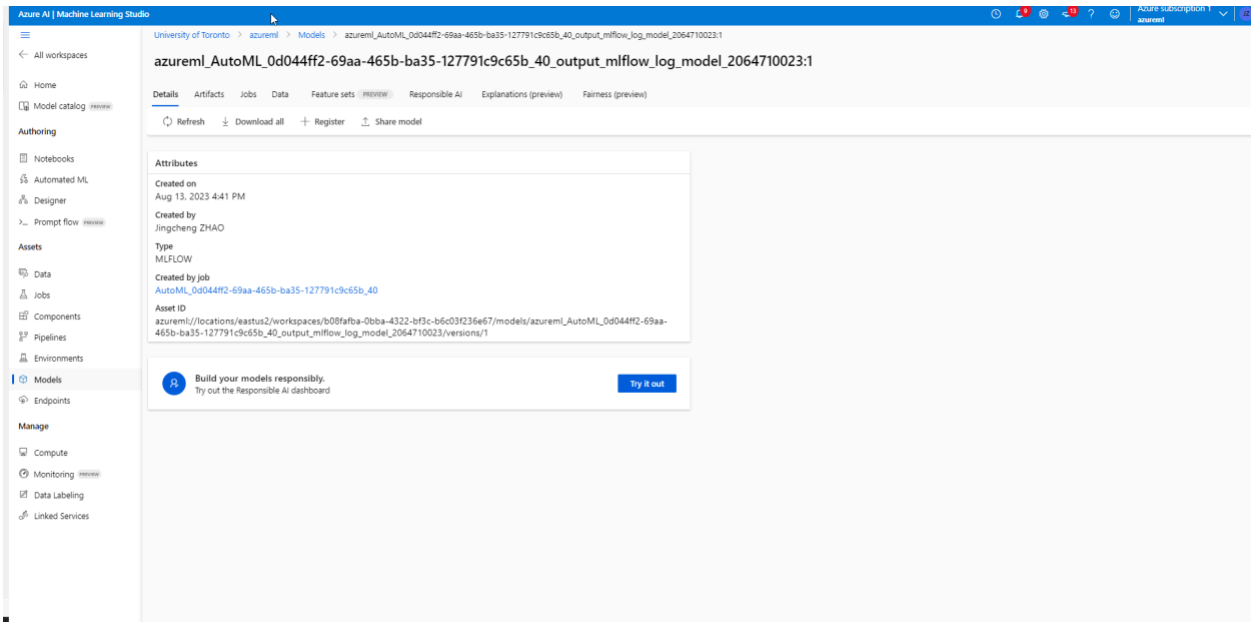


Automated ML Results:

Results Overview:

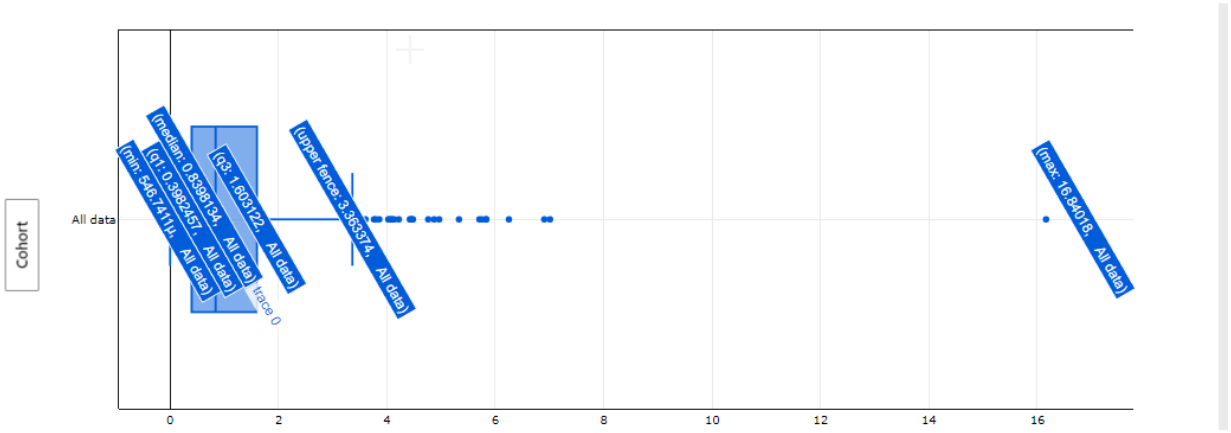


Best Model:

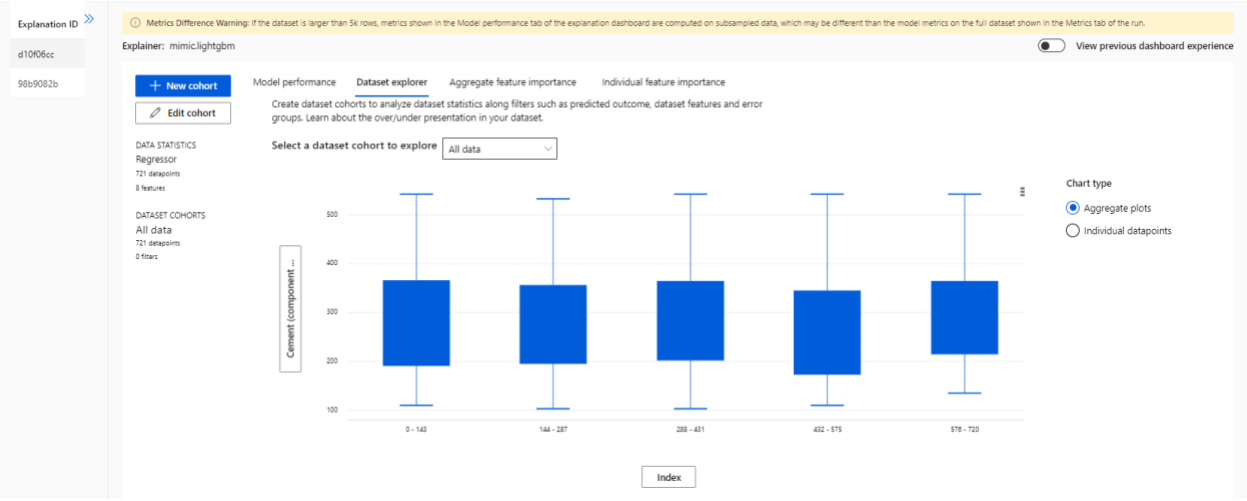


Ensemble Details:

Best Model Performance:

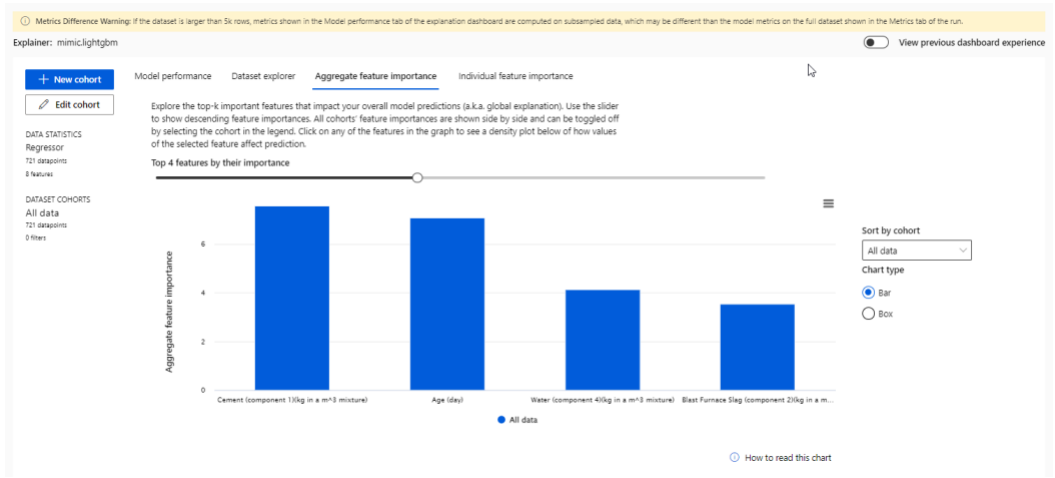


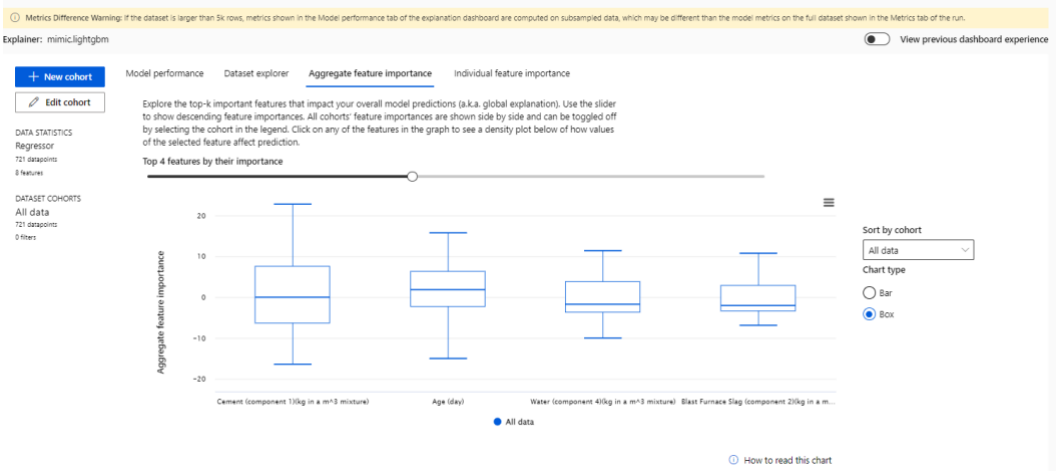
Dataset Explorer: (Aggregate plots)



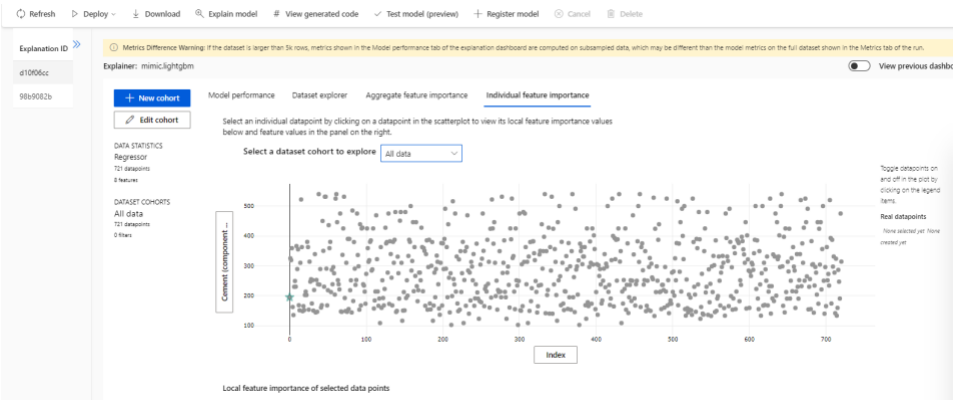
Dataset Explorer: (Individual datapoints)

Feature Importance:



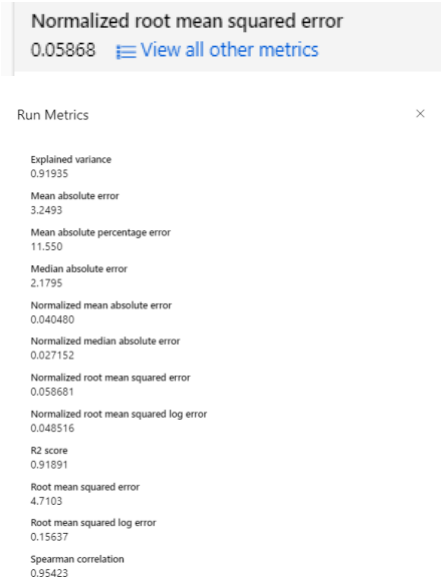


Individual feature importance:

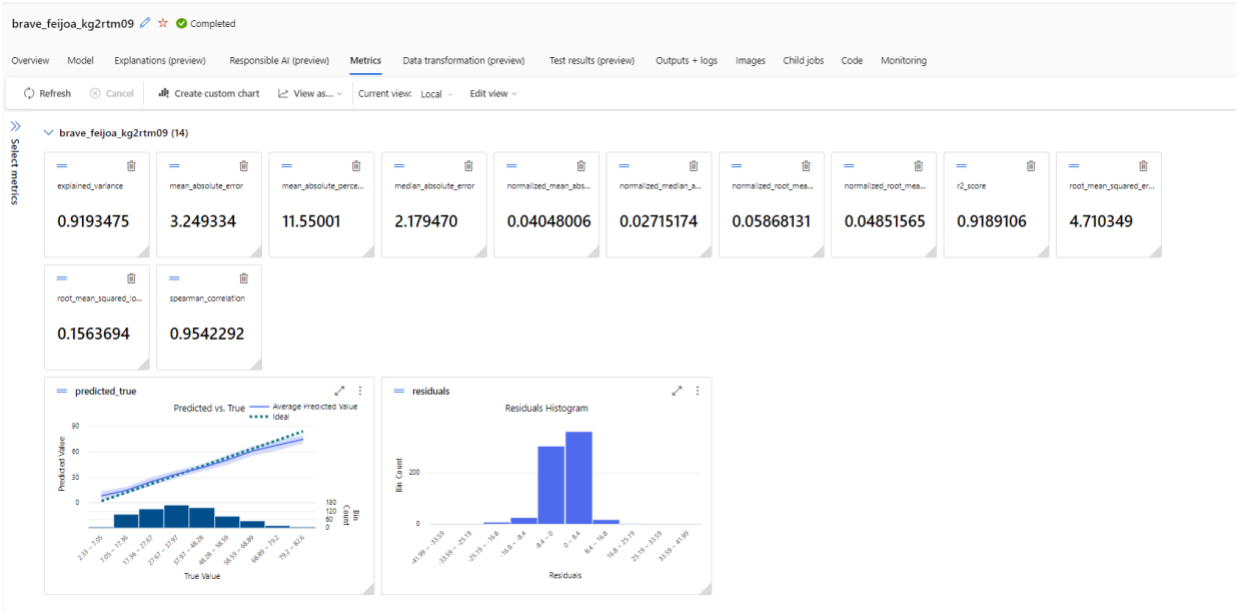


Data Transformation:

Metrics Results:



Final Results Visualization:



Best model results explanation:

The best model algorithm is **Voting Ensemble**. Explained Variance is 0.9193. A higher explained variance indicates that the model's predictions closely match the actual target values. In this case, 0.9193 is quite high, suggesting that the model's predictions capture a substantial amount of variance in the target variable. The mean absolute error is 3.249, a lower MAE indicates that, on average, the model's predictions are close to the actual values. In this case, 3.249 is not very ideal compared to the 2 models we trained in question4. The model's RMSE is 4.71, not excellent but it indicates that the model's predictions are, on average, closer to the actual values. The R-squared value of this model is 0.91891, which is very good since it's very close to 1. A higher R-squared value (close to 1) indicates that the model's predictions closely match the actual variability in the data. A lower MAPE indicates that, on average, the model's predictions have a smaller percentage error. In this case, the average percentage error is 11.55%.

Based on the metrics, the **Voting Ensemble** model seems to perform well. It captures a significant portion of the variance in the target variable, has low errors, and indicates a strong fit to the data. It's essential to consider the context of your problem and domain-specific requirements when interpreting these results. Meanwhile, as we can see in the predicted vs true line graph, the prediction line is very close to the true value line. However, the performance of the 2 models we trained in question4 is even better in terms of RMSE and MAE, which means there is still room for improvement in Microsoft Azure Automated ML.