

Demand Forecasting for Seoul Bike System using Machine learning

Ensemble Multivariable Regressions - Integrating Azure Machine Learning and Azure Auto ML

The dataset that I am working with is called “**Seoul Bike Sharing Demand**” which represents the number of public bikes rented every hour for every day from December 2017 to end of November 2018 (1 year of data) for Seoul’s bike sharing system. The target variable of the dataset is the column “**Rented Bike Count**” which states the number of bikes rented in that hour. There are 13 features that are provided with the dataset, with time-based features such as “Date” “Hour” and “Seasons” and weather based features such as “Temperature”, “Humidity” and “Visibility”.

The problem and the goal of addressing this problem, and why it is meaningful:

The problem addressed is one of demand forecasting. As I am looking to enter the field of supply chain optimization, this project is particularly important to my long-term career goals. The goal of this project is accurately predicting demand for number of bikes required at different hours of the day based on historical data of environmental factors, temporal patterns and cultural factors to smoothen the supply of bikes.

A break down for why this problem is important and meaningful to address:

By Accurate Demand Forecasting the following aspects can be optimized

1. **Prevent Demand Supply Mismatch:** Prevent lost revenue from unmet demand and costs from having over allocating bikes which is wasted opportunity cost and additional maintenance cost.
2. **Enhance operational efficiency:** Provides clear path and structure for bike allotment (smoothen supply allotment process of bikes) and prevents unnecessary redistribution of bikes to meet demand from warehouses.
3. **Financial Decision Making:** Allows Seoul Bike service to create accurate budgets and decisions for optimal fleet size, distribution planning and labor allotment which allows to save costs for improving other aspects.
4. **Customer Satisfaction:** Allows better user experience by ensuring that there is low waiting time and met demand for customers, promoting customer loyalty and future business.

A crucial assumption is that the target variable called “Rented Bike Count” is the required demand for bikes at that hour. This assumption is based because we do not have information in the dataset whether there exists a supply side constraint at any given hour. If there is a supply side constraint, or inefficiencies in allotment, there does exist the possibility that the demand predictions may be underpredicted. However, given the absence of supply-side information in this dataset, using the “Rented Bike Count” as a proxy for demand is a reasonable assumption.

Data Cleaning

1. Dataset contains no missing values and therefore no data imputation, or deletion of rows/columns are required.
2. Dataset contains no duplicate rows, so dealing with this issue not required.
3. Column feature called “Functioning Day” and all associate rows where this column has the value “No” will be removed.
4. Lastly, we will remove the feature “Date” from our dataset.

Data Pre-Processing

1. The hour column is cyclical in nature and will be engineered to 2 features to represent this, after which the hour column will be deleted.
2. The column “Seasons” is categorical and nominal, and so will be encoded with one-hot encoding.
3. “Holiday” is categorical and binary, and therefore will be encoded as 1/0 column.
4. Continuous variable features will be standardized manually to prepare for training and validation sets.

Modelling Approach

1. The problem will be modeled as regression problem with the target being the demand (“Rented Bike Count”) - depending on features such as time of day, environment features and holidays.
2. Azure ML studio with Azure Notebooks and common Python libraries will be utilized to perform cleaning, preprocessing and then EDA and visualizations. Furthermore, we will split data into train (75%), and (25%) on validation to get the performance of our ML algorithms on the dataset. .
3. I will be utilizing Random Forest regression and Linear Regression with default parameters as our 2 algorithms.
4. To check model performance, we will be using performance measures such as RMSE, MAE and R^2 value.
5. Feature importances will be output from each model to understand feature-target relationships.
6. Lastly, I will utilize “Automated ML” with regression in Azure to test combinations of relevant algorithms and hyperparameters. This will be run for 300 minutes to get comprehensive model testing. The best model will be analyzed and tested on untouched test data.

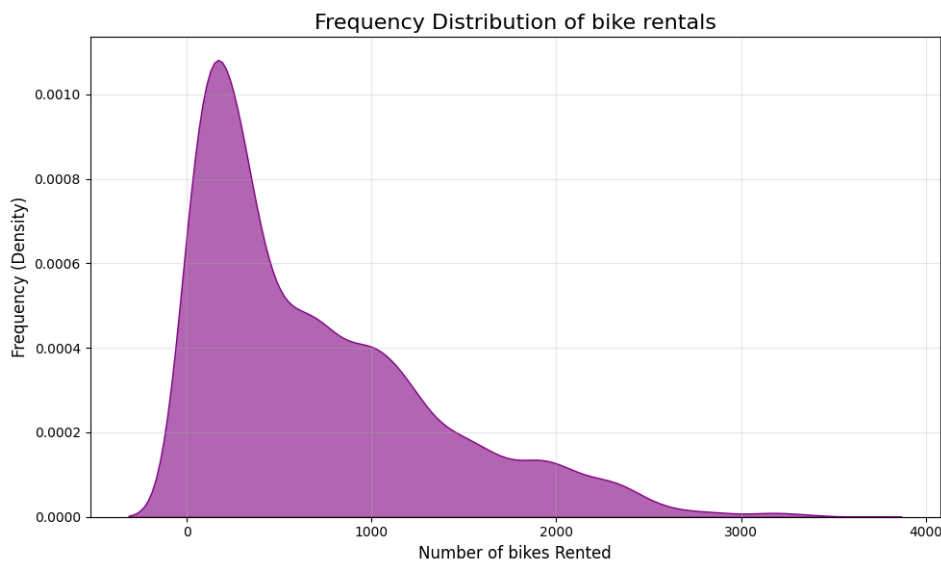
Jupyter Notebook Environment in Azure Machine Learning

The screenshot displays the Azure Machine Learning Studio interface. The left sidebar shows the 'Notebooks' section with a file explorer. The main area shows a Jupyter Notebook titled 'neil_1628_a5_part8.ipynb' running on a 'neil-a5-part8-computr' instance. The code in the notebook imports pandas, seaborn, matplotlib, and sklearn, and reads data from 'SeoulBikeData.csv'. The output shows a preview of the data with columns: Date, Rented Bike Count, Hour, Temperature(°C), Humidity(%), Wind speed (m/s), Visibility (10m), Dew point temperature(°C), Solar Radiation (MJ/m2), Rainfall(mm), Snowfall (cm), Seasons, Holiday, and Fur.

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)	Seasons	Holiday	Fur
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	Winter	No Holiday	
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	Winter	No Holiday	

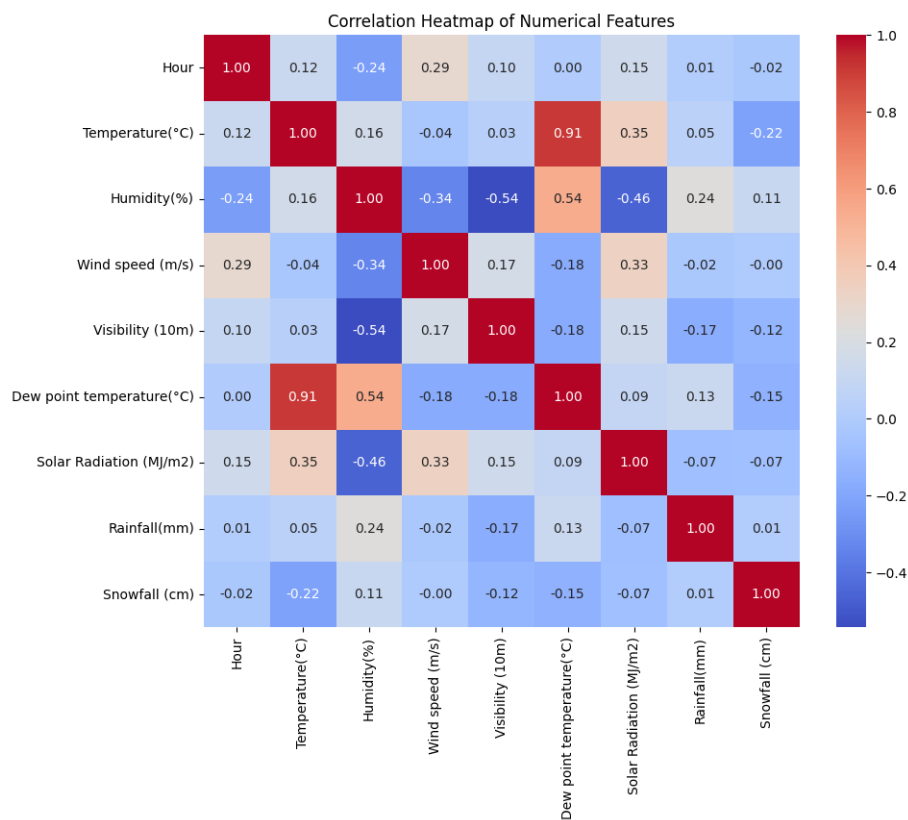
Exploratory Data Analysis

Graph 1:



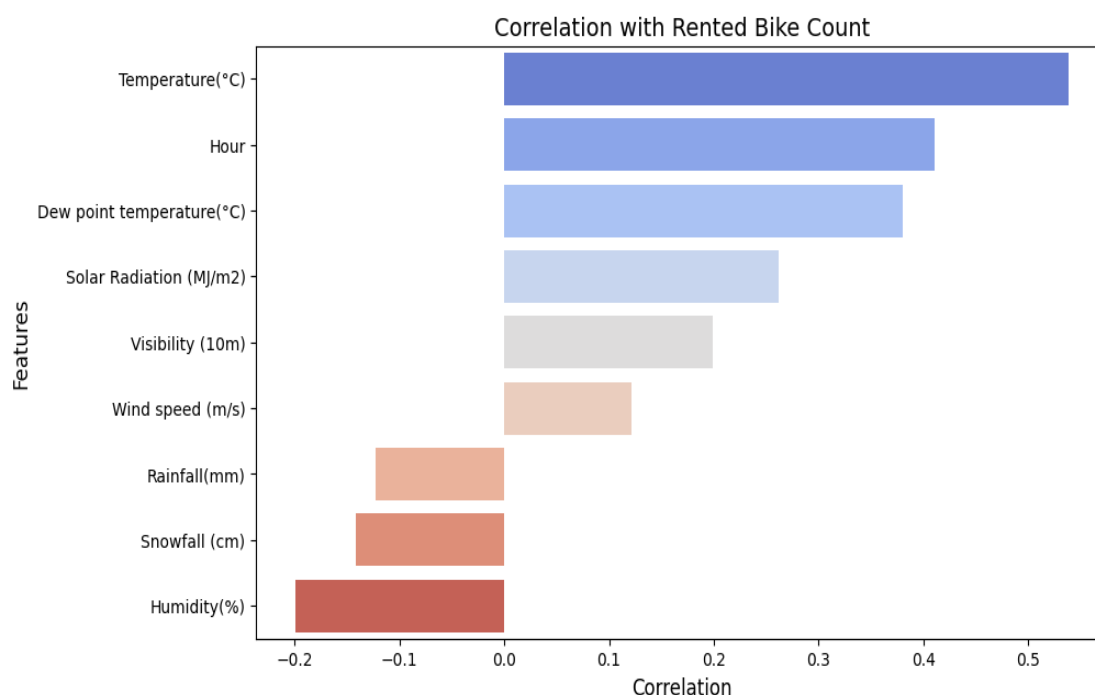
Explanation: This graph shows frequency density distribution of bike rentals per hour, indicating that most bike rentals are concentrated at lower values of around 100-1000 bikes rented. The distribution is right-skewed, with comparatively lesser frequency density after 2000 bikes.

Graph 2:



Explanation: This heatmap reveals correlations between numerical features which ranges from positive (+1) to negative (-1). Most features have very low correlation with each other as shown by the values and red and blue colour darkness, meaning each feature captures unique information. It is meaningful as it helps identify feature relationships and potential multicollinearity for better feature understanding and selection if required.

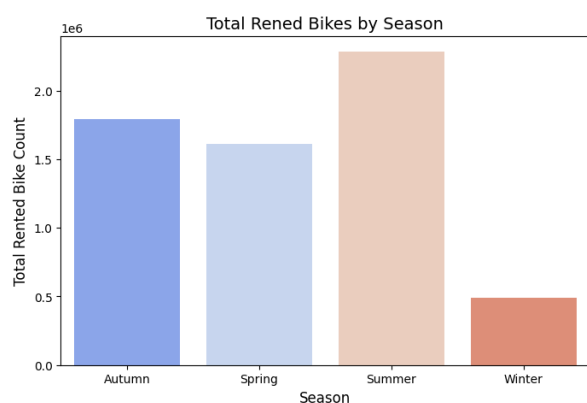
Graph 3:



Explanation: This is a correlation bar plot that exhibits the relationship between the different numerical features and the target variable of rented bikes every hour (between -1 and 1). Temperature has a strong positive correlation (increase correlated to increase in rented bikes) with approximately 0.55, followed by hour of day. Humidity (-0.2) followed by snowfall have negative correlation. (Decrease in these values leads to increase in rented bikes). These results make qualitative sense.

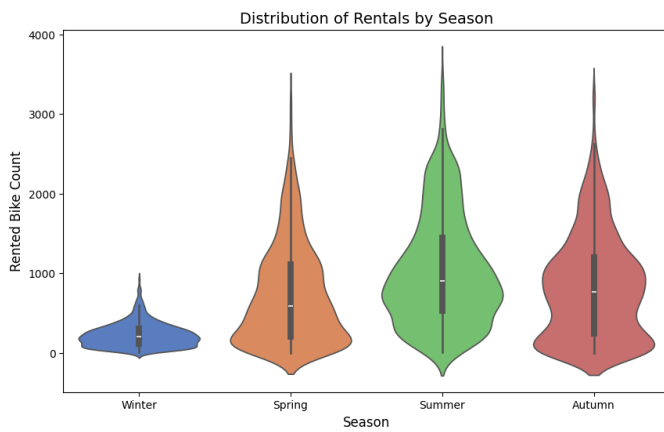
Graph 4

Output Graph:



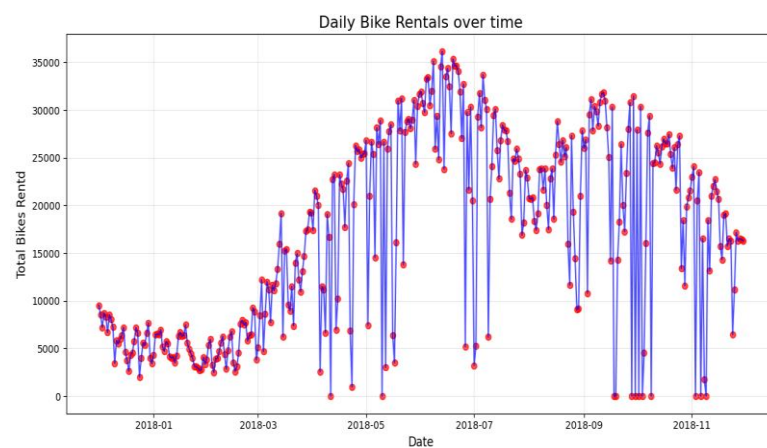
Explanation: This is a bar plot that sums the total number of bikes that have been rented out by 4 different seasons: Autumn, Spring, Summer and Winter. We saw earlier that the number of data points across all seasons are nearly the same, so this provides an accurate distribution. We see that in the summer the greatest number of bikes have been sold, followed by Autumn, and with Winter having significantly less, which makes sense because of poor weather conditions.

Graph 5:



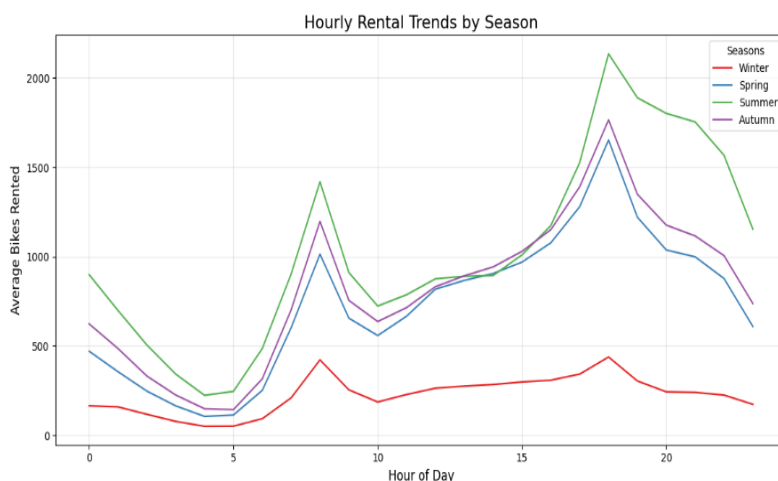
This violin plot shows the distribution and density of bike rentals by season. Summer has the highest density around 800 rentals, while winter has the lowest and most concentrated rentals. Spring and autumn show more spread-out distributions with notable outliers, indicating higher variability compared to winter.

Graph 6:



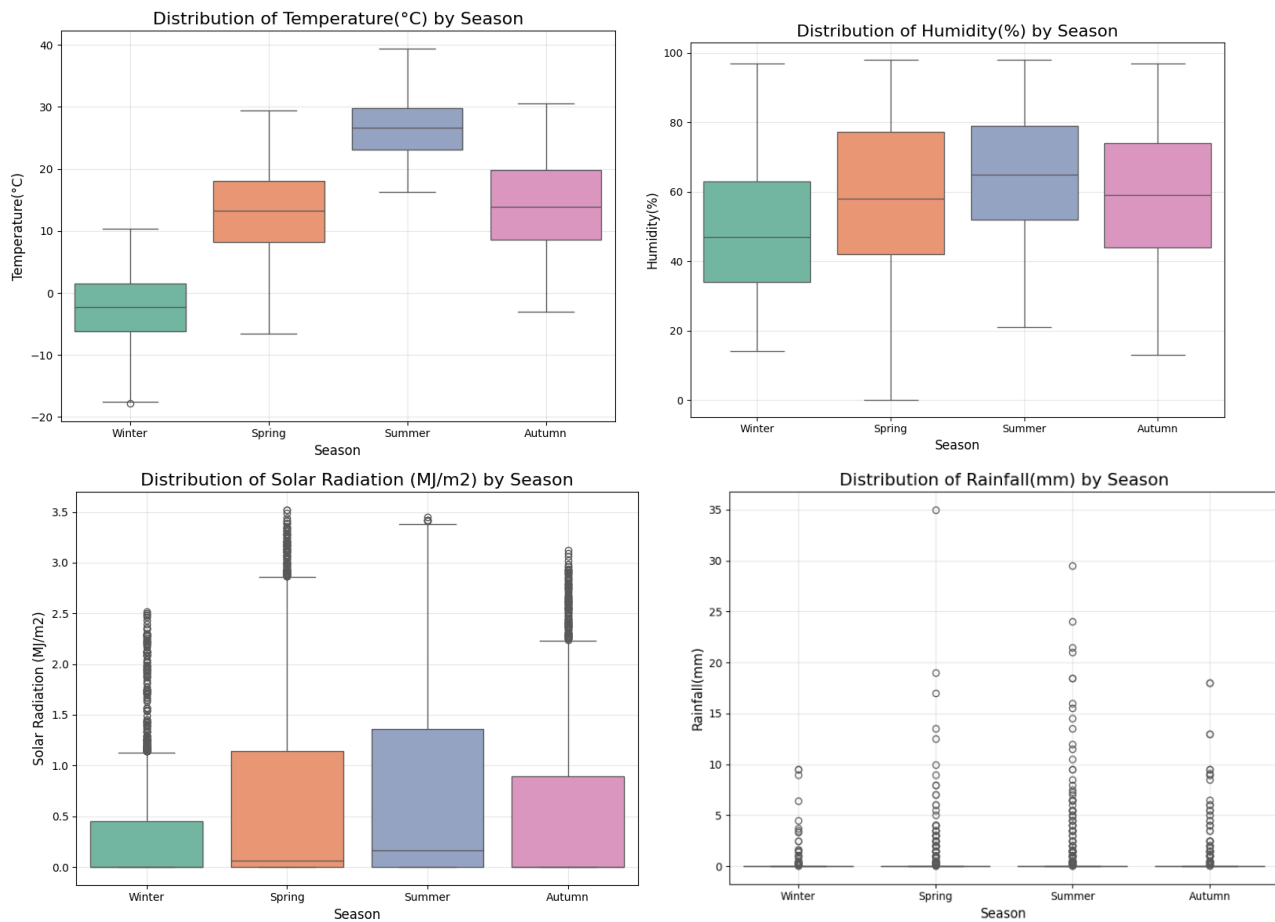
Explanation: This graph sums the number of bikes rented per day for the entire year. It is a time series plotted with a line and scatter plot. A general trend is observed where the bikes rented increase during days of April to July, and then falls significantly in November to March which matches the trend of the seasons that we observed. This graph also shows that the increases and decreases are not smooth, as we see frequent troughs, suggesting some disruptions or extreme conditions leading to drop.

Graph 7:



Explanation: This line plot demonstrates the average bikes rented every hour of the day. There are 4 plots, where the average each hours calculated for each different season. It is observed that the peak hours of bikes being rented is 8 AM and 6 PM, which are usually peak times for commute. Generally higher trends of rented bikes are observed in the afternoon and evening compared to night and early mornings. It is interesting to see that the trend of average bikes rented every hour does not change by the season. i.e. those are still the peak times for each season.

Graph Outputs:



Explanation: The 4 plots show the box plot distributions of different values of environmental factors (temperature, humidity, Solar Radiation, and Rainfall) for each of the 4 different seasons. For temperature, we see that the summer has the highest mean temperatures, followed by Autumn. Humidity is very similar across autumn, spring and summer, but with outliers being similar and present for all 4 seasons, but Spring has the most spread of humidity values. For Solar Radiation values, winter has the most number of outliers, but spring has a good amount of outliers but with many peaks in solar radiation. Summer has almost no outliers and is low in terms of Solar radiation for extremes. For rainfall, all four seasons have most days with no rain. But spring has the highest value of rainfall, while summer has the most number of outliers for rainfall. These graphs provide a good idea how these environmental factors distribute by season.

Data Cleaning

Justifications

1. Firstly, we have no missing values and duplicates in our dataset as shown in the first two lines of the output. Therefore, it is not required to deal with imputation, or deletions to handle these values.
2. Next step of cleaning process was to remove the column called 'Functioning Day', and all the associated rows where this value was 'No'. This is because this represents when the bike system did not work at all, and hence all bikes rented during these days are 0. If we had kept these rows, it would have biased the results and impacted other features predictions:

```
1 func_no = df[df['Functioning Day'] == 'No']
2 func_no['Rented Bike Count'].value_counts()
3
```

[21] ✓ <1 sec

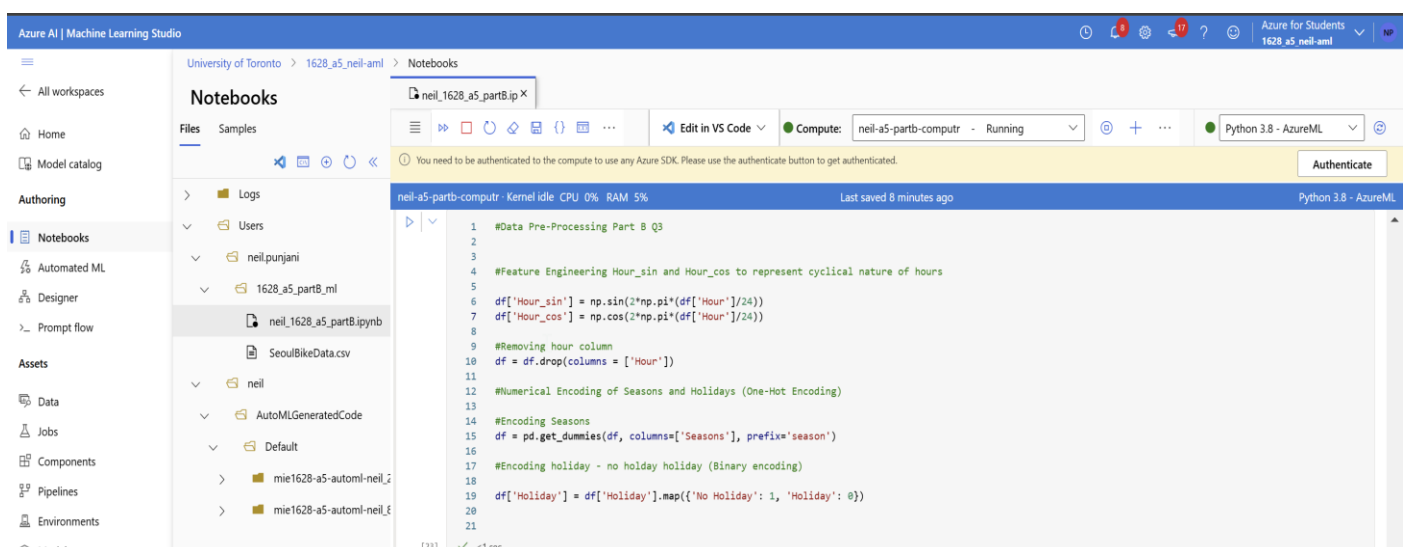
... 0 295

Name: Rented Bike Count, dtype: int64

3. Last step was to remove the date column from our features set. This was done because of these reasons:
 - As we have information such as Seasons, holidays and other environmental factors, the inherent information that the dates capture is already captured by these other features.
 - As this is a date series, our model would have to switch to time series forecasting, which would require us to remove the rest of our features for prediction and rely entirely on this. Integrating both increases complexity, dimensionality and increases redundancy. It would make sense to use time series if this was the only feature.
 - Lastly, we only have data from 1 year. Therefore, it is hard to capture direct seasonality from this data, and wont be very helpful for long term predictions, and with just one year of data can lead to inaccuracy.

Data Processing

Code:



The screenshot shows the Azure AI Machine Learning Studio interface. On the left is a sidebar with navigation options like 'All workspaces', 'Home', 'Model catalog', 'Authoring', 'Assets', and 'Models'. The main area displays a Jupyter Notebook titled 'neil_1628_a5_partB.ipynb'. The notebook content includes the following code:

```
1 #Data Pre-Processing Part B Q3
2
3
4 #Feature Engineering Hour_sin and Hour_cos to represent cyclical nature of hours
5
6 df['Hour_sin'] = np.sin(2*np.pi*(df['Hour']/24))
7 df['Hour_cos'] = np.cos(2*np.pi*(df['Hour']/24))
8
9 #Removing hour column
10 df = df.drop(columns = ['Hour'])
11
12 #Numerical Encoding of Seasons and Holidays (One-Hot Encoding)
13
14 #Encoding Seasons
15 df = pd.get_dummies(df, columns=['Seasons'], prefix='season')
16
17 #Encoding holiday - no holiday holiday (Binary encoding)
18
19 df['Holiday'] = df['Holiday'].map({'No Holiday': 1, 'Holiday': 0})
20
21
```

The output of the last cell is shown as [21] ✓ <1 sec.

Justification:

1. The hour column is one that represents periodic time frames from 0-23. Therefore, hour 23 is close to 0, but the model might interpret it as being further way from each other if we keep the feature as such,

which would impact the model performance. Therefore, I engineered two features to convert this information to cyclical encoding (conversion to radians and then sin and cos of this radian are 2 columns) to ensure that the relationship between the times is preserved. The hour column is then removed to eliminate redundancy.

Reference: <https://shrmtmt.medium.com/understand-the-capabilities-of-cyclic-encoding-5b68f831387e>

2. Next, we have columns of seasons, which have 4 values. This is a nominal categorical variable where there is no inherent ranking between the seasons. Therefore, to encode this, I performed one-hot encoding, which engineered 4 binary features for each season where an entry represents if that data row belongs to that season (1), or if it doesn't (0).
3. Next, we needed to encode the binary categorical variable called Holiday Column which has either a holiday or no holiday for that day. This is simple as the values associated with No Holiday were encoded as 1, and if there was a Holiday, it was encoded as 0.
4. Lastly, the columns associated with numerical values were standardized with StandardScaler. This was done so to ensure that our linear regression model converges effectively and would also give us interpretable feature importance coefficients. It was ensured that the binary encoded and one-hot encoded columns were not standardized, otherwise they would lose meaning and interpretability.

Model Testing

Model 1: Linear Regression

Data was split into **75% for training and 25% for validation** post standardization.

For both models, I chose the following performance measures:

1. RMSE: This would allow us to see how far our predictions are from the target, with values further away being penalized more. RMSE also brings error to same scale as target, making it more interpretable.
2. MAE: I chose this to see the absolute errors on average for a more general idea for performance. Combined with RMSE, it can help understand presence of major outliers in predictions.
3. R^2 : I chose this performance measure as it captures how much the model fit well for capturing the variance, making it interpretable as the value is usually between 0 and 1, providing a benchmark.

The first model that I selected was a Linear Regression model to fit the data. I chose this because.

1. Linear regression model is a good baseline model. If a more complex model performs worse than it, it can signify overfitting or unnecessary complexity that should be avoided. This would also help me digest the relationship a bit easier and see if non-linear patterns are present.
2. Linear regression trains models very quickly and allows immediate results. For our dataset with approximately 8500 rows of data, it can help with a quicker first pass on the data.
3. Linear regression is easy to understand and interpretable. The coefficients directly show how features influence the target variable, helping us relate their importance and impact.
4. We have many continuous variables for our features, and linear regression can handle them well.

Code:

```

1 #Linear regression model fit and error calculation
2 model_1 = LinearRegression()
3 model_1.fit(X_train, y_train)
4 y_pred_lr = model_1.predict(X_val)
5
6 rmse_lr = root_mean_squared_error(y_val, y_pred_lr)
7 mae_lr = mean_absolute_error(y_val, y_pred_lr)
8 r2_lr = r2_score(y_val, y_pred_lr)
9
10 print(f"R-squared (R²): {r2_lr:.2f}")
11 print(f"RMSE: {rmse_lr:.2f}")
12 print(f"MAE: {mae_lr:.2f}")

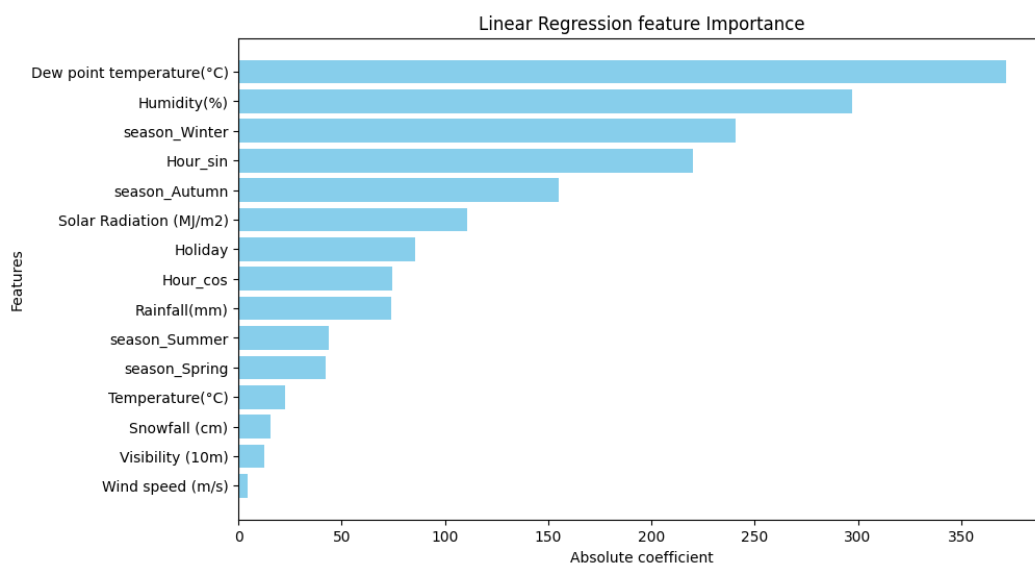
```

Output:

```

R-squared (R²): 0.53
RMSE: 436.29
MAE: 324.45

```



The linear regression model is trained with default parameters. After being trained, It performs quite poorly on the validation data. We receive **RMSE value of 436.29** and **MAE value of 324.45** (On our scale is high), with an **R² value of 0.53**. This indicates that our model is not a good fit for this dataset, and there are non linear relationships that exist between the features the target, which our linear regression model cannot capture. This can be addressed by performing feature engineering, or hyperparameter tuning to include regularization, or trying another ML algorithm that can capture non-linearity, which is what we will do.

The feature important plot shows that the **dew point temperature** has highest coefficient in predicting the target, followed by **humidity**. The lowest feature importances are **visibility** and **windspeed**. As the model is not performing very well, we cannot be confident in these importances, so we can compare after we train another model.

Model 2: Random Forest Regressor

Why did I choose this model?

- As we have nonlinearity between features and target, Random Forest Regressor is an ideal algorithm to capture these trends effectively.
- It will be able to capture feature interactions without feature engineering which can lead to improved model performance.
- As it averages results from multiple trees, it will mitigate risk of overfitting. As we have slightly high number of features for our data, this is will help a lot.
- It will be resistant to outliers in our target, as it averages results in different partitions.

- It can handle both categorical and numerical data effectively, and as we have both of them, it will help.
- It can also output feature importances, which will help us understand the underlying most important features that impact the rented bikes.

Code:

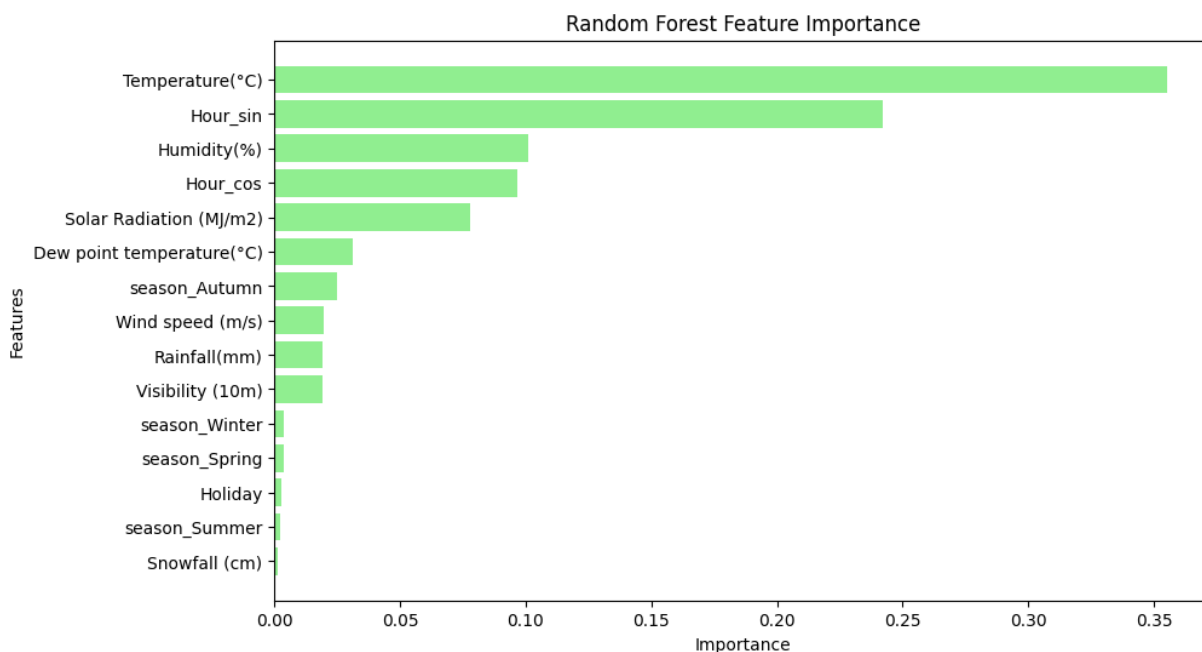
The screenshot shows the Azure Machine Learning Studio interface. On the left is a sidebar with navigation options like 'All workspaces', 'Home', 'Model catalog', 'Authoring', 'Notebooks', 'Automated ML', 'Designer', 'Prompt flow', 'Assets', 'Data', 'Jobs', and 'Components'. The main area displays a Jupyter Notebook titled 'neil_1628_a5_part8.ipynb'. The code in the notebook is as follows:

```
1 #Random forest fit and error calculation
2 model_2 = RandomForestRegressor()
3 model_2.fit(X_train, y_train)
4 y_pred_rf = model_2.predict(X_val)
5
6 rmse_rf = root_mean_squared_error(y_val, y_pred_rf)
7 mae_rf = mean_absolute_error(y_val, y_pred_rf)
8 r2_rf = r2_score(y_val, y_pred_rf)
9
10 print(f"R-squared (R²): {r2_rf:.2f}")
11 print(f"RMSE: {rmse_rf:.2f}")
12 print(f"MAE: {mae_rf:.2f}")
```

The output of the code is displayed below the cell:

```
[24]: ✓ 4 sec
R-squared (R²): 0.84
RMSE: 254.02
MAE: 153.01
```

Feature Importance:



The Random Forest model is trained with default parameters. The model performs quite well in terms of its performance measures on the validation set. It has **RMSE of 254.02, MAE of 153.01, and R² of 0.84**. This model performs significantly better than the linear regression model, so our inference that there was nonlinear relationships is confirmed. It has low error on RMSE and MAE. The RMSE being higher than MAE tell us that there are some big outliers in terms of error in predictions, but the low MAE says that on average, the model is doing well. Hyperparameter tuning could enhance this model.

Lastly, we see that for feature importances, the most important is **Temperature**, followed by **hour_sin**. Compared with our old correlation plots, it matches up, because a colder temperature is likely to put off people from taking the bikes. While as we saw in EDA, there is higher distribution in summer so this adds up as well. Next, the hour of the day significantly impacts as we saw in EDA that there were specific peaks when demand rose. It tells us that time of day is also significantly important. Due to a higher R² and lower error, these feature importances are more reliable and dependable.

Q5.

After it was understood that significant non-linear patterns were present in the dataset, I utilized Azure Auto ML for further hyperparameter tuning and exploration of ensemble regression models. To begin the model, the first step was to output the processed data from Azure ML notebook. This processed data was constructed post processing and cleaning with one hot encoded and binary encoded variables and seasonal engineer features, but without standardization of continuous columns.

Auto ML automatically handles standardization of continuous variables, and often treats it as a hyperparameter, so pre standardization was avoided. This data was outputted using the following code in Azure ML notebook post processing:

Creating Data Asset in Auto ML job with this dataset, and then reading it for use in our auto ml job.

Create data asset

Settings
These settings determine how the data is parsed. The initial settings are automatically detected; you can change them as needed to reparse the data.

File format: Delimited | Delimiter: Comma | Example: Field1,Field2,Field3 | Encoding: UTF-8

Column headers: All files have same headers | Skip rows: None

☐ Dataset contains multi-line data

Note: Processing tabular files with multi-line data is slower because multiple CPU cores cannot be used to ingest the data in parallel. Checking this option may result in slower processing times.

Data preview

Rented ...	Temper...	Humidi...	Wind s...	Visibilit...	Dew po...	Solar R...	Rainfall...	Snowfa...	Holiday	Hour_sin	Hour_cos	season...
254	-5.2	37	2.2	2000	-17.6	0	0	0	1	0	1	0
204	-5.5	38	0.8	2000	-17.6	0	0	0	1	0.259	0.966	0
173	-6	39	1	2000	-17.7	0	0	0	1	0.5	0.866	0
107	-6.2	40	0.9	2000	-17.6	0	0	0	1	0.707	0.707	0
78	-6	36	2.3	2000	-18.6	0	0	0	1	0.866	0.5	0
100	-6.4	37	1.5	2000	-18.7	0	0	0	1	0.966	0.259	0
181	-6.6	35	1.3	2000	-19.5	0	0	0	1	1	0	0

Back Next Review Cancel

1.1 Auto ML Job Submission Summary

Submit a training job

Review
Review or make changes to your job before submission.

Basic settings

Name: mie1628-a5-automi-neil

Experiment name: Default

Timeout (hours): --

Task type & data

Task type: Regression

Data: --

Task settings

Target column: Rented Bike Count

Limits

- Max trials: 1000
- Max concurrent trials: 1
- Max nodes: --
- Metric score threshold: --
- Experiment timeout (minutes): 300
- Iteration timeout (minutes): 15

Validate type: Automatic

Test data: Train-test split

Percentage test of data: 25

Compute settings

Compute target: a5-1628-cc-neil

Back Submit training job Cancel

5.5 Job was run for 5 hours (300 minutes). The screenshot below show status of Auto ML job completion.

Properties

- Status: Completed
- Script name: --
- Created by: Neil Sanjiv Punjani
- Job type: Automated ML
- Experiment: Default
- Arguments: None
- See all properties
- Raw JSON
- See YAML job definition
- Job YAML

Inputs

- Input name: training_data
- Data asset: SeoulBikeDataAuto-ML-1
- Asset URI: azureml:SeoulBikeDataAuto-ML-1

Outputs

- Output name: best_model
- Model: azureml_mie1628-a5-automl-neil_250_output_mlflow_log_model_1083123288:1
- Asset URI: azureml:azureml_mie1628-a5-automl-neil_250_output_mlflow_log_model_1083123288:1
- Output name: full_training_dataset
- Dataset: bd622469-0551-4674-8fba-0d29a35f87fb

Best model summary

- Algorithm name: VotingEnsemble
- Ensemble details: View ensemble details

A total of 254 different regression models were run by auto ml. The below screenshots show the top 10 performing models by their normalized RMSE:

Algorithm name	Explained	Responsible AI	Normalized ro...	Sampling	Created on	Duration	Hyperparameter
VotingEnsemble	View explanation		0.06208	100.00 %	Dec 6, 2024 6:45 AM	1m 23s	algorithm : [LightGBM, LightGBM] ...
MaxAbsScaler, LightGBM			0.06243	100.00 %	Dec 6, 2024 1:45 AM	27s	min_data_in_leaf : 20 ...
StandardScalerWrapper, LightGBM			0.06527	100.00 %	Dec 6, 2024 6:22 AM	46s	boosting_type : gbdt colsampl...
MaxAbsScaler, XGBoostRegressor			0.06643	100.00 %	Dec 6, 2024 1:45 AM	28s	tree_method : auto ...
MinMaxScaler, RandomForest			0.06868	100.00 %	Dec 6, 2024 3:07 AM	43s	bootstrap criterion : squared_e...
MinMaxScaler, RandomForest			0.07028	100.00 %	Dec 6, 2024 1:45 AM	26s	bootstrap criterion : squared_e...
MinMaxScaler, RandomForest			0.07040	100.00 %	Dec 6, 2024 4:51 AM	44s	bootstrap : true criterion : squ...
StandardScalerWrapper, RandomForest			0.07129	100.00 %	Dec 6, 2024 3:34 AM	43s	bootstrap criterion : squared_e...
MinMaxScaler, RandomForest			0.07279	100.00 %	Dec 6, 2024 4:08 AM	44s	bootstrap criterion : squared_e...
MaxAbsScaler, ExtremeRandomTrees			0.07324	100.00 %	Dec 6, 2024 1:45 AM	25s	bootstrap criterion : squared_e...

What is the Best Model?

Based on all the models tested, the best model is an ensemble model called **“Voting Ensemble”**. This model incorporates 4 different regression algorithms. These 4 algorithms are **LightGBM with Standard Scaler Wrapper**, **XGBoostRegressor with Sparse Normalizer**, **XGBoostRegressor with MaxAbsScaler**, and **LightGBM with MaxAbsScaler**. They have weights 0.133, 0.067, 0.133 and 0.667 respectively.

This ensemble model works by weighing the predictions of each model by that specific weight and then summing them to get the predicted value for that data point. This ensemble model helps because it reduces impacts of errors from individual models. It also allows leveraging strengths of different models to get performance. The screenshot below shows:

Ensemble details

Select an ensemble algorithm to see the ensemble weights and hyperparameters.

- ☐ StandardScalerWrapper, LightGBM
- ☐ SparseNormalizer, XGBoostRegressor
- ☐ MaxAbsScaler, XGBoostRegressor
- ☒ MaxAbsScaler, LightGBM

Ensemble weight: 0.6666666666666666

Data transformation:

```

1 {
2   "spec_class": "preproc",
3   "class_name":
4     "MaxAbsScaler",
5   "module": "sklearn.
6     preprocessing",
7   "param_args": {},
8   "param_kwargs": {},
9   "prepared_kwargs": {}
10 }

```

Training algorithm:

```

1 {
2   "spec_class": "sklearn",
3   "class_name":
4     "LightGBMRegressor",
5   "module": "automl.
6     client.core.common.
7     model_wrappers",
8   "param_args": {},
9   "param_kwargs": {
10     "min_data_in_leaf":
11       20
12   },
13   "prepared_kwargs": {}
14 }

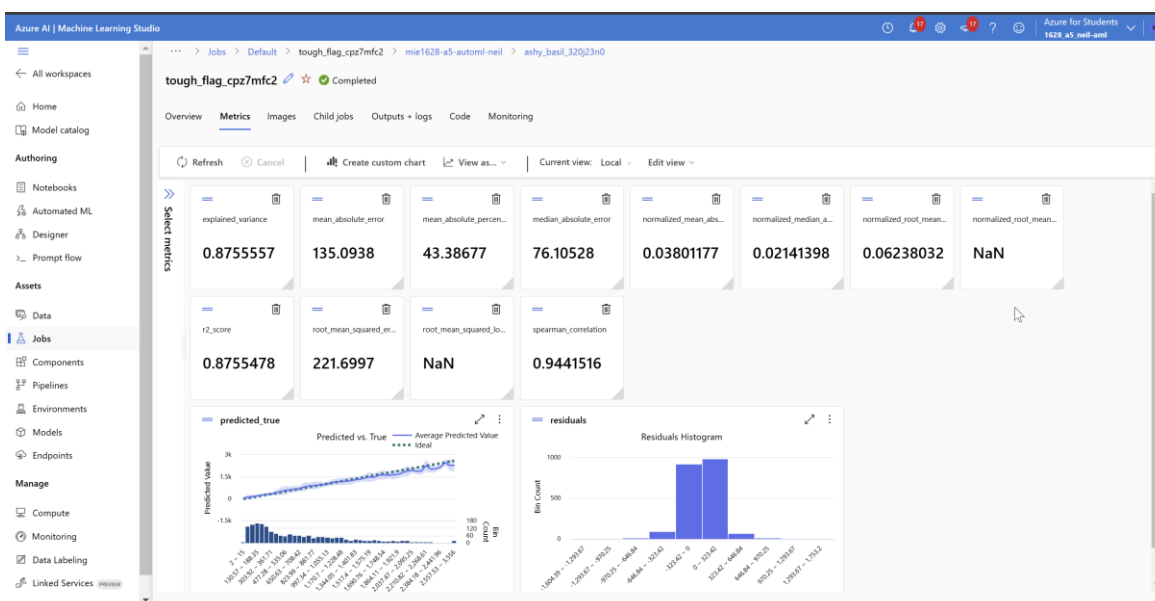
```

Explanation of Best Model:

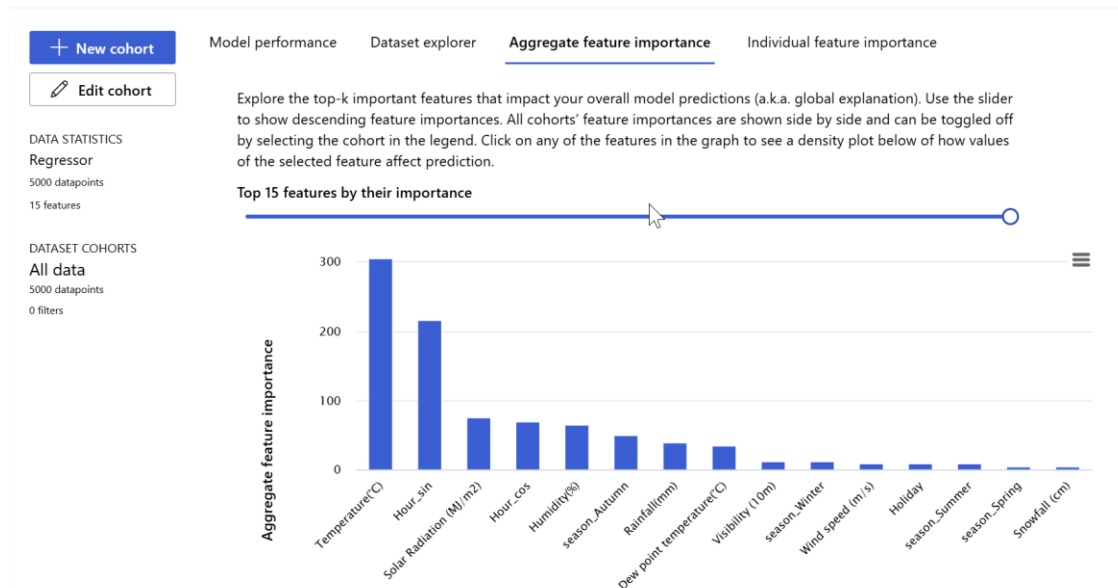
The best model performs well on validation data. We receive an R^2 value of 0.88, MAE of 142.08 and RMSE of 220.62. This is a better performing model than the random forest model we trained earlier. However with 5 hours of training and trying 254 models, we were able to up the performance a slight bit but not drastically, which could suggest that there is inherent randomness, inaccurate data points or the features we have are not sufficient to accurately model the non linear relationships. Some potential methods of improvement and future research are suggested in Part B Q6.

Furthermore, the model was tested on unseen 25% of test data that was untouched for the training process. This resulted in R^2 of 0.88, MAE of 135.1 and RMSE of 221.7, which is extremely similar to the model's performance on validation set. This means model is performing well to completely unseen data, suggesting low overfitting, and proper generalization of algorithm.

For feature importances, the results are very similar to the feature importances observed in the Random Forest Model with temperature and hour_sin being the same ranking, and other importances rank being close to each other. This gives us confirmed evidence that temperature and hour of day are the most important features for predicting the Bike demand every hour, and snowfall being the least important



Best Model Feature Importances:



Q6.

With this project, there was a successful demonstration of the application of machine learning for the task of demand forecasting of the number of bikes required every hour for the Seoul bike-sharing system. A strong **R² value of 0.88**, **MAE of 135.1**, and **RMSE of 221.7** was achieved. This best model was obtained from Azure's Auto ML job function with the **Voting Ensemble** model, which utilized four different models, with **LightGBM Max Scaler** having the highest weight. These strong results support the ability to model demand for bikes using temporal, environmental, and categorical features. From our models, **temperature** and **hour_sin** of the day were identified as the most important features. If this model is utilized in real-world scenarios, improved demand forecasting could lead to increased revenue, enhanced customer satisfaction, and better allotment and fleet decisions.

While improvements over baseline models like **linear regression** and **random forest** were observed with AutoML, the slight performance gain after training for 5 hours and testing 254 models suggests inherent limitations such as noise, insufficient features, or non-linear relationships that were not fully captured. To further improve model performance and address these limitations, manual hyperparameter tuning on the top 10 models could be performed. Furthermore, during the Auto ML job, one option not utilized was **stacking ensemble models**, which could capture relationships by training a meta-model based on individual model outputs, potentially improving performance over simpler ensemble models. Future research could also explore integrating advanced algorithms like **CatBoost** or **deep learning** and leveraging external data, such as traffic patterns or event schedules, to refine demand forecasting performance and enhance operational planning.

Reference for stacking ensemble: <https://learn.microsoft.com/en-us/azure/machine-learning/concept-automated-ml?view=azureml-api-2#ensemble>