CAPITAL BIKESHARING DEMAND PREDICTION

MUHAMMAD SULTAN PASYA

CONTEXT

Capital Bikeshare (CaBi) serves Washington, D.C., and parts of its metropolitan area, with over 700 stations and 5,400 bikes as of January 2023. These automated bike rentals enhance urban mobility, reduce environmental impact, and promote health. Since launching in 2010, CaBi has generated rich data on user behavior and traffic patterns, offering insights valuable for urban planning and transportation.

PROBLEM STATEMENT

How to predict bike demand to improve station operations and balance bike availability throughout the day?

GOALS

Develop a model to predict bike rentals based on weather, temperature, and time, allowing for optimized operations, balanced bike distribution, and improved service efficiency.

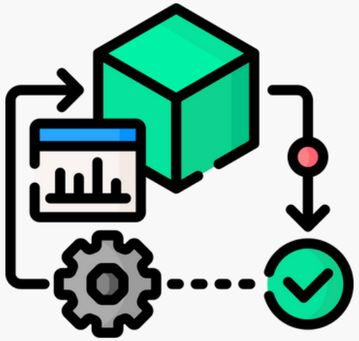
ANALYTICAL APPROACH: REGRESSION

DATA UNDERSTANDING

FEATURE SELECTION & DATA SPLITTING

DATA PREPROCESSING

MODELLING



CONCLUSION & RECOMMENDATIONS

EVALUATION METRICS

RMSE

Measures the square root of the average squared differences between predicted and actual values. Lower RMSE indicates better model performance.

MAE

Calculates the average of the absolute differences between predicted and actual values. A lower MAE suggests more accurate predictions.

MAPE

Expresses prediction errors as a percentage of actual values. Lower MAPE indicates better relative accuracy in model predictions.

DATA UNDERSTANDING

DATA INFORMATION

	dteday	hum	weathersit	holiday	season	atemp	temp	hr	casual	registered	cnt
0	2011-12-09	0.62	1	0	4	0.3485	0.36	16	24	226	250
1	2012-06-17	0.64	1	0	2	0.5152	0.54	4	2	16	18
2	2011-06-15	0.53	1	0	2	0.6212	0.62	23	17	90	107
3	2012-03-31	0.87	2	0	2	0.3485	0.36	8	19	126	145
4	2012-07-31	0.55	1	0	3	0.6970	0.76	18	99	758	857
...	...	...	...	...	...	...	...	...	...	...	...
12160	2012-01-25	0.75	1	0	1	0.2273	0.24	7	14	243	257
12161	2012-07-06	0.62	1	0	3	0.7424	0.78	0	39	63	102
12162	2012-02-20	0.60	2	1	1	0.2121	0.24	5	0	6	6
12163	2012-03-31	0.77	2	0	2	0.4242	0.42	2	14	55	69
12164	2011-04-28	0.47	1	0	2	0.6212	0.64	18	44	486	530

12165 rows × 11 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12165 entries, 0 to 12164
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   dteday      12165 non-null  object
 1   hum         12165 non-null  float64
 2   weathersit  12165 non-null  int64
 3   holiday     12165 non-null  int64
 4   season      12165 non-null  int64
 5   atemp       12165 non-null  float64
 6   temp        12165 non-null  float64
 7   hr          12165 non-null  int64
 8   casual      12165 non-null  int64
 9   registered  12165 non-null  int64
 10  cnt         12165 non-null  int64
dtypes: float64(3), int64(7), object(1)
memory usage: 1.0+ MB
```

Initial dataset contains 12,165 rows and 11 columns: 10 numerics and 1 (dteday) in object format. We will convert dteday to the proper datetime format.

DATA DICTIONARY

Feature	Description
dteday	Date
season	Season (1: winter, 2: spring, 3: summer, 4: fall)
hr	Hour (0 to 23)
holiday	Whether the day is a holiday (0: No, 1: Yes)
temp	Normalized temperature in Celsius. Values derived via `(t-tmin)/(tmax-tmin)` where tmin=-8°C, tmax=+39°C (Hourly scale)
atemp	Normalized feeling temperature in Celsius. Values derived via `(t-tmin)/(tmax-tmin)` where tmin=-16°C, tmax=+50°C (Hourly scale)
hum	Normalized humidity. Values divided by 100 (maximum)
casual	Count of casual users
registered	Count of registered users
cnt	Count of total rental bikes including both casual and registered users
weathersit	Weather situation:
	1: Clear, Few clouds, Partly cloudy
	2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds
	3: Light Snow, Light Rain + Thunderstorm + Scattered clouds
	4: Heavy Rain + Ice Pellets + Thunderstorm + Mist, Snow + Fog

Each row represents the total bike rental bookings for a specific hour on a given date, including features like date, season, hour, holiday status, temperatures, humidity, and weather situation.

CONVERTING DATA AND RENAMING COLUMNS

```
df["year"] = df["dteday"].dt.year
df["month"] = df["dteday"].dt.month.astype("category")
df["day"] = df["dteday"].dt.day_name().astype("category")
```

✓ 0.0s

```
# Rename the columns to make it easier to understand
df = df.rename(columns={
    "hum": "humidity",
    "weathersit": "weather",
    "atemp": "feel_temp",
    "hr": "hour",
    "cnt": "count"
})
df.head()
```

✓ 0.0s

To simplify processing, we'll split the dteday column into separate year, month, and day columns. Since casual and registered counts are captured in cnt, we'll delete these columns. We'll also rename some columns for easier reference in future analysis.

ANOMALIES AND DUPLICATED DATA

	humidity
count	12165.000000
mean	0.625451
std	0.192102
min	0.000000
25%	0.470000
50%	0.620000
75%	0.780000
max	1.000000

```
df.duplicated().sum()
```

✓ 0.0s

```
1
```

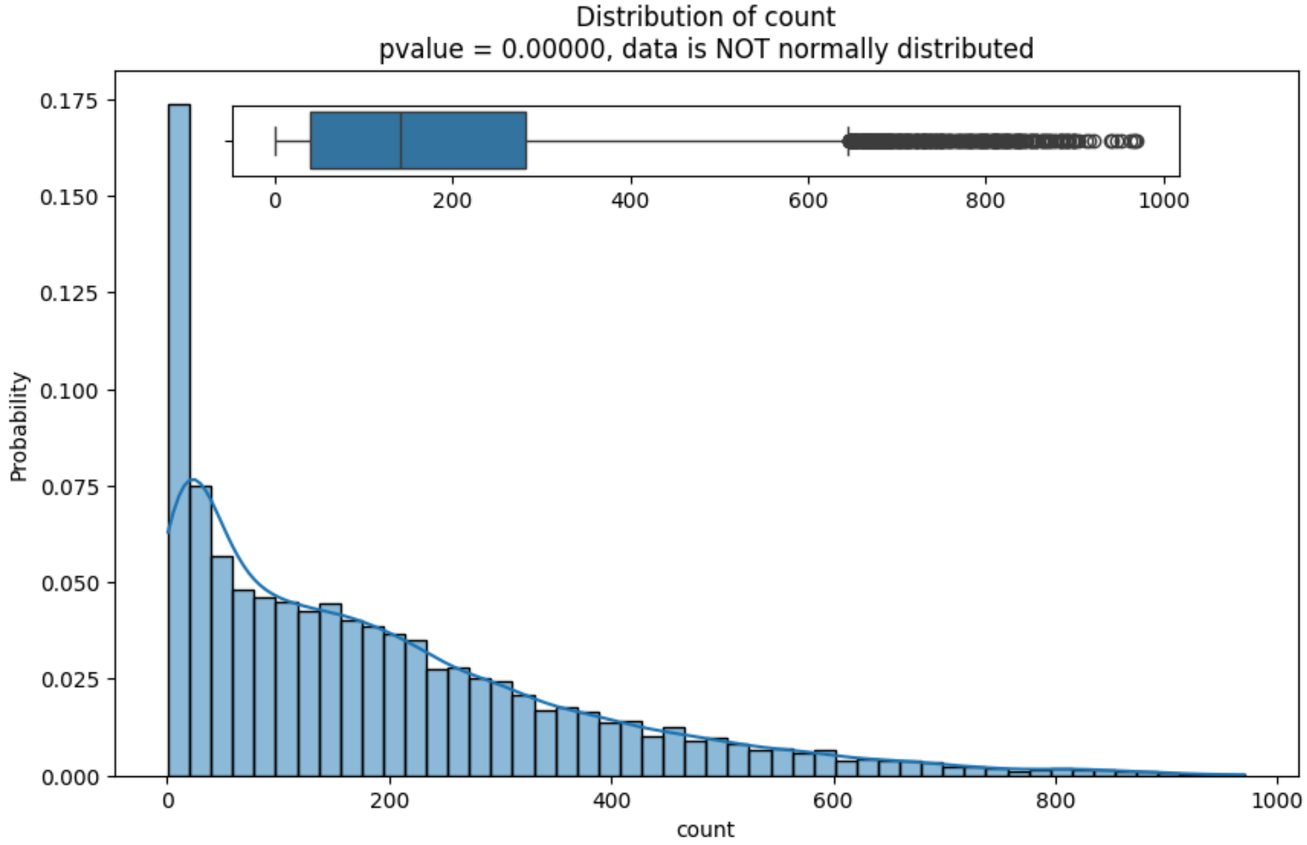
There is one anomaly with a humidity value of 0, which is impossible, as well as one duplicate entry. Therefore, we will remove both the anomaly and the duplicate data.

FINAL DATASET

```
<class 'pandas.core.frame.DataFrame'>
Index: 12150 entries, 0 to 12164
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   humidity   12150 non-null  float64
 1   weather    12150 non-null  int64
 2   holiday    12150 non-null  int64
 3   season     12150 non-null  int64
 4   feel_temp  12150 non-null  float64
 5   temp       12150 non-null  float64
 6   hour       12150 non-null  int64
 7   count      12150 non-null  int64
 8   year       12150 non-null  int32
 9   month      12150 non-null  category
 10  day        12150 non-null  category
dtypes: category(2), float64(3), int32(1), int64(5)
memory usage: 926.1 KB
```

After renaming some columns, deleting unnecessary columns, cleaning the anomalies, and dropping duplicated data, we have a total of 12150 rows and 11 columns.

DATA DISTRIBUTION



The count variable is heavily right-skewed, indicating most rentals are low. To improve model performance and reduce the influence of extreme values, we will apply a transformation to normalize the count variable.

FEATURE SELECTION & DATA SPLITTING

X AND Y

In this analysis,

- the features (X) will include: all columns except count,
- which will serve as our target (y).

This setup ensures the model predicts based solely on the other variables.

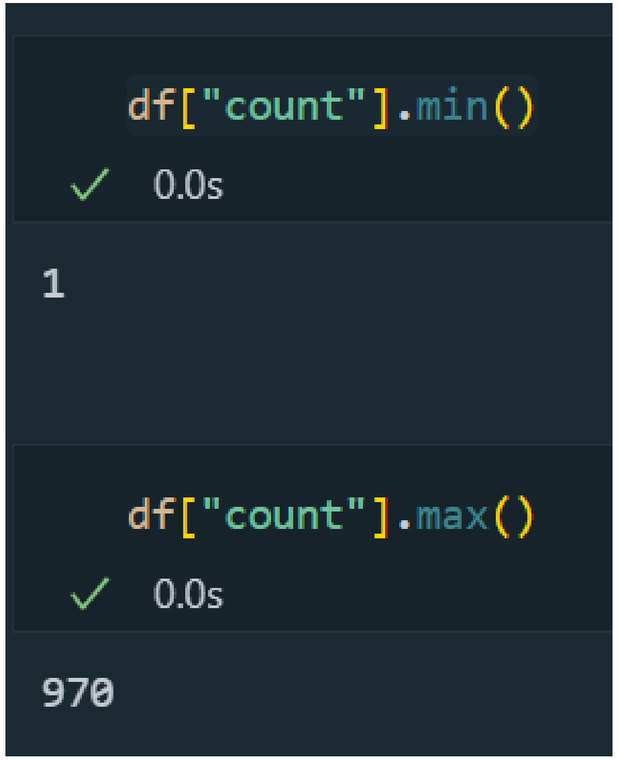
DATA SPLITTING

The data is split into training and testing sets for X and y using an 80:20 ratio. The split will create:

- X_train: training features
- X_test: testing features
- y_train: training target
- y_test: testing target

This allows us to train the model on 80% of the data and test its performance on 20%.

VALUE RANGE



The target variable has values ranging from 1 to 970. This range is important for interpreting the Root Mean Square Error (RMSE) metric.

DATA PREPROCESSING

BINNING

We will apply ordinal binning with 5 uniform bins to the following continuous features:

- `humidity`
- `feel_temp`
- `temp`

SCALING

We will use RobustScaler to scale the following features to ensure they are within a specific range:

- `weather`
- `season`
- `hour`
- `year`
- `month`
- `day`

ENCODING

For categorical variables, we will use OneHotEncoder to transform:

- `month`
- `day`

PIPELINE

Since 'month' and 'day' will be encoded and scaled, we will create a pipeline for these two columns.

COLUMN TRANSFORMER

```
transformer = ColumnTransformer(
    [("Hum, Temp, Feel", KBinsDiscretizer(n_bins=5, encode="ordinal", strategy="uniform"), ["humidity", "temp", "feel_temp"]),
     ("OH_Rob", onehot_robust, ["day", "month"]),
     ("Robust", RobustScaler(), ["weather", "season", "hour", "year"])],
     remainder="passthrough"
)
```

This is the final data preprocessing step,
featuring the Column Transformer I used.

DATA MODELLING

CROSS-VALIDATION SETUP

MODELS TO COMPARE

Base Models:

- Linear Regression
- K-Nearest Neighbors
- Decision Tree

Ensemble Models:

- Random Forest
- AdaBoost
- XGBoost

SCORING METRICS

Since we are dealing with regression data, we will use the following scoring metrics:

- RMSE
- MAE
- MAPE

CROSS-VALIDATION

KFold with n_splits = 5, shuffle = True, random_state = 19

CROSS-VALIDATION RESULT

	Mean RMSE	STD RMSE	Mean MAE	STD Mae	Mean MAPE	STD MAPE
XGB	-45.329082	2.938431	-27.506795	1.207153	-0.256465	0.007049
Random Forest	-50.540100	3.445591	-30.193541	1.093237	-0.294278	0.007704
Decision Tree	-65.311067	3.570057	-38.515474	1.148234	-0.389983	0.015113
KNN	-139.074309	3.566504	-88.256607	1.697032	-1.448803	0.070099
ADA Boost	-147.600433	3.049535	-88.698085	1.889025	-0.530100	0.012964
Linear Regression	-165.869010	1.669989	-109.368733	1.416509	-1.430169	0.036368

Based on the cross-validation results, we found that Extreme Gradient Boost (XGB) exhibited the least error by a significant margin.

EXTREME GRADIENT BOOST

XGBoost is an optimized implementation of the gradient boosting algorithm designed for performance and speed. It builds an ensemble of decision trees sequentially, where each tree attempts to correct the errors of the previous ones by minimizing a differentiable loss function.

Example

Suppose you want to predict house prices based on features like size, number of rooms, and location. XGBoost starts with an initial guess (e.g., the average price). In the first iteration, it builds a decision tree to predict the residual (the difference between the predicted and actual prices). Then, in subsequent iterations, more trees are built to minimize the residuals until the final model gives an accurate prediction of house prices.

HYPERPARAMETER TUNING

We will optimize the XGBoost model by tuning the following parameters to improve accuracy and generalization:

- **max_depth: Maximum tree depth (controls overfitting).**
- **learning_rate: Step size for weight updates (affects accuracy).**
- **n_estimators: Number of trees (more trees can improve accuracy).**
- **subsample: Fraction of data used for trees (adds randomness to reduce overfitting).**
- **gamma: Minimum loss reduction for splitting (prevents unnecessary splits).**
- **colsample_bytree: Fraction of features for each tree (reduces overfitting).**
- **reg_alpha: L1 regularization on weights (helps prevent overfitting).**

This tuning will enhance model performance on the test data.

HYPERPARAMETER TUNING RESULTS

```
hyperparam = {
    "modeling__regressor__max_depth": list(np.arange(1, 11)),
    "modeling__regressor__learning_rate": list(np.arange(0.1, 1, 0.1).round(2)),
    "modeling__regressor__n_estimators": list(np.arange(100, 501, 10)),
    "modeling__regressor__subsample": list(np.arange(1, 11)/10),
    "modeling__regressor__gamma": list(np.arange(0, 11)),
    "modeling__regressor__colsample_bytree": list(np.arange(1, 10)/10),
    "modeling__regressor__reg_alpha": list(np.logspace(-3, 3, 7))
}
```

	RMSE	MAE	MAPE
XGB	41.644678	25.299804	0.243471
XGB_1	39.954534	24.269413	0.243220

After predicting with the original XGBoost (XGB) model on the test data, we saw improvements across all metrics. However, the MAPE showed only slight enhancement, indicating room for further optimization. To address this, we will conduct another round of hyperparameter tuning on the tuned model (XGB_1) to refine performance and reduce percentage errors significantly.

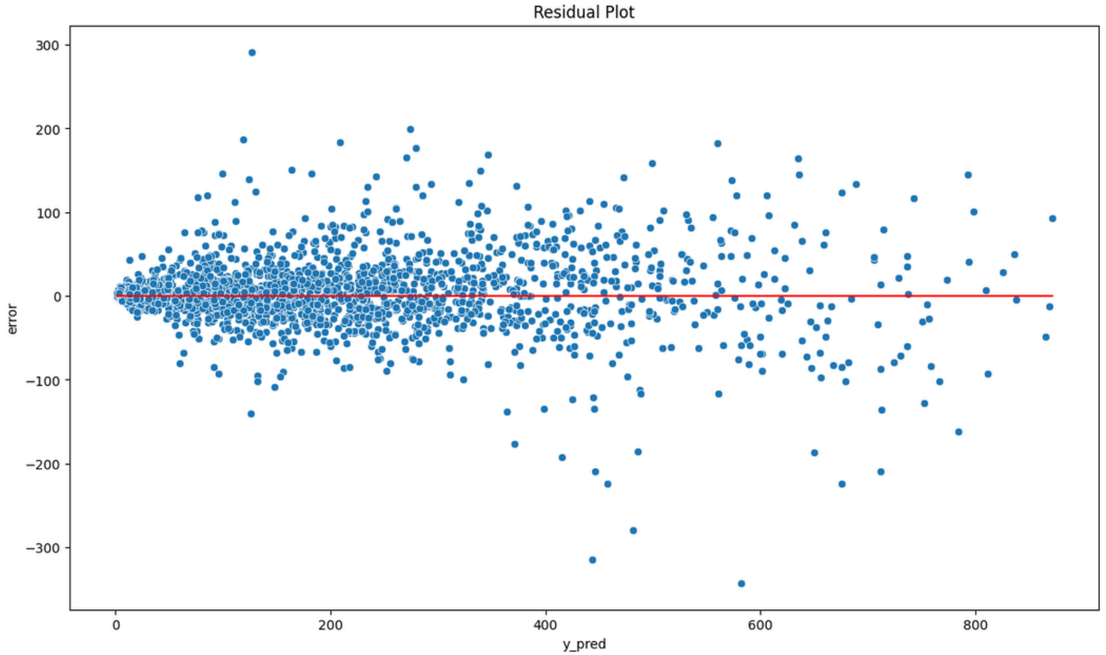
HYPERPARAMETER TUNING RESULTS

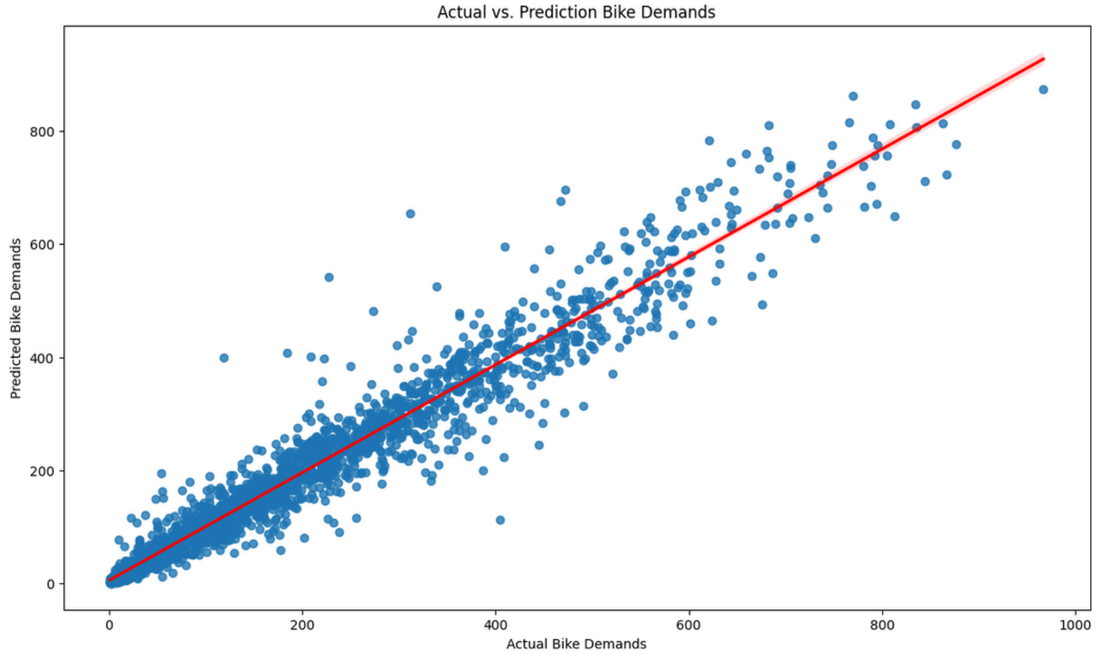
```
hyperparam_2 = {
    "modeling__regressor__max_depth": list(np.arange(6, 11)),
    "modeling__regressor__learning_rate": list(np.arange(0.01, 0.31, 0.05).round(2)),
    "modeling__regressor__n_estimators": list(np.arange(100, 701, 50)),
    "modeling__regressor__subsample": list(np.arange(0.01, 0.31, 0.01).round(2)),
    "modeling__regressor__colsample_bytree": list(np.arange(0.5, 1.0, 0.1).round(2)),
    "modeling__regressor__reg_alpha": list(np.logspace(-3, 1, 5))
}
```

	RMSE	MAE	MAPE
XGB	41.644678	25.299804	0.243471
XGB_1	39.954534	24.269413	0.243220
XGB_2	40.128083	24.394314	0.242351

After the second round of tuning, the first tuned model (XGB_1) shows the best overall performance, with the lowest RMSE (39.95) and MAE (24.27), indicating accurate predictions with smaller errors. While the second tuned model (XGB_2) slightly excels in MAPE, it only improves slightly. Therefore, considering the RMSE and MAE metrics, XGB_1 is the most reliable model for accurate predictions.

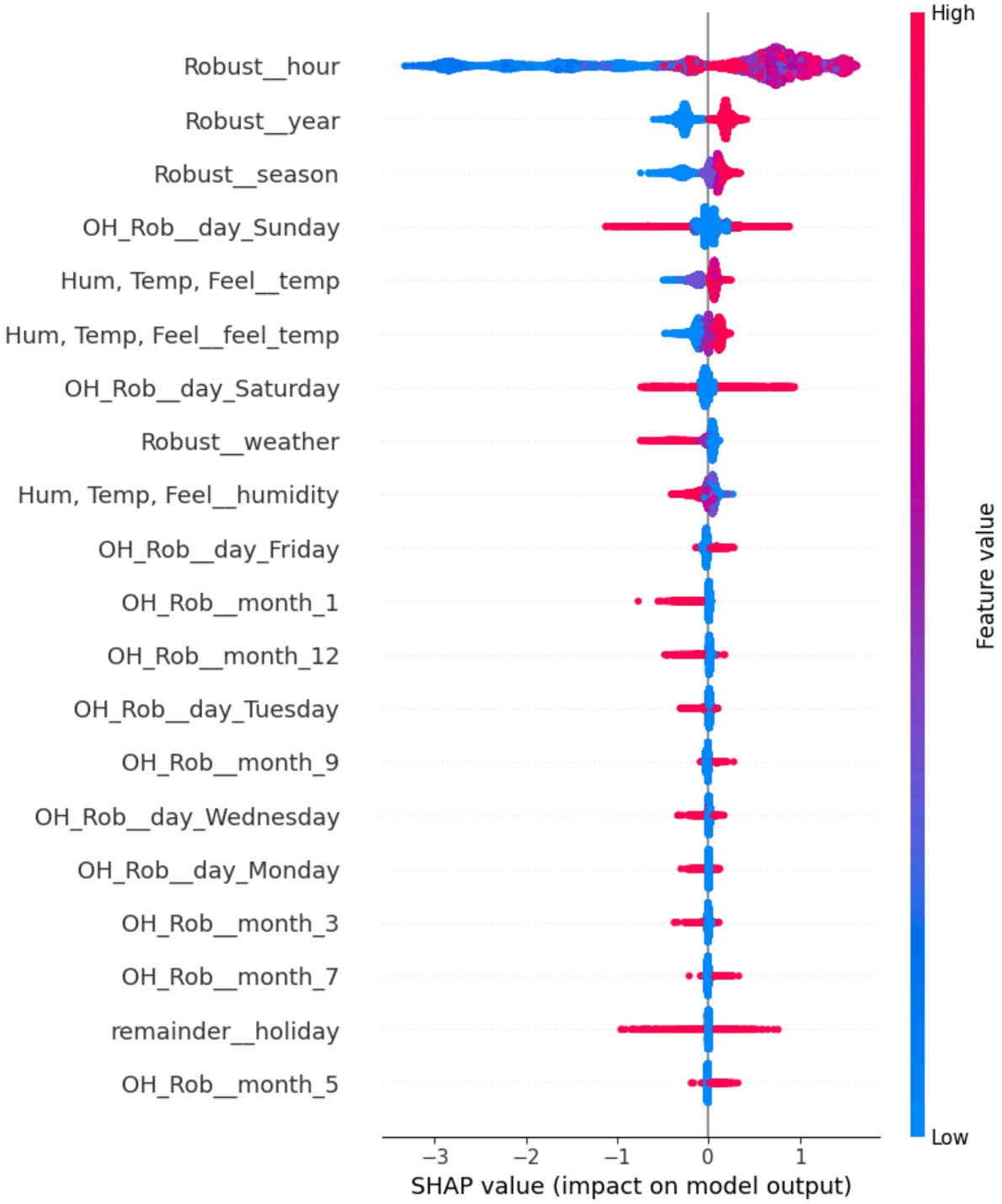
MODEL EVALUATION



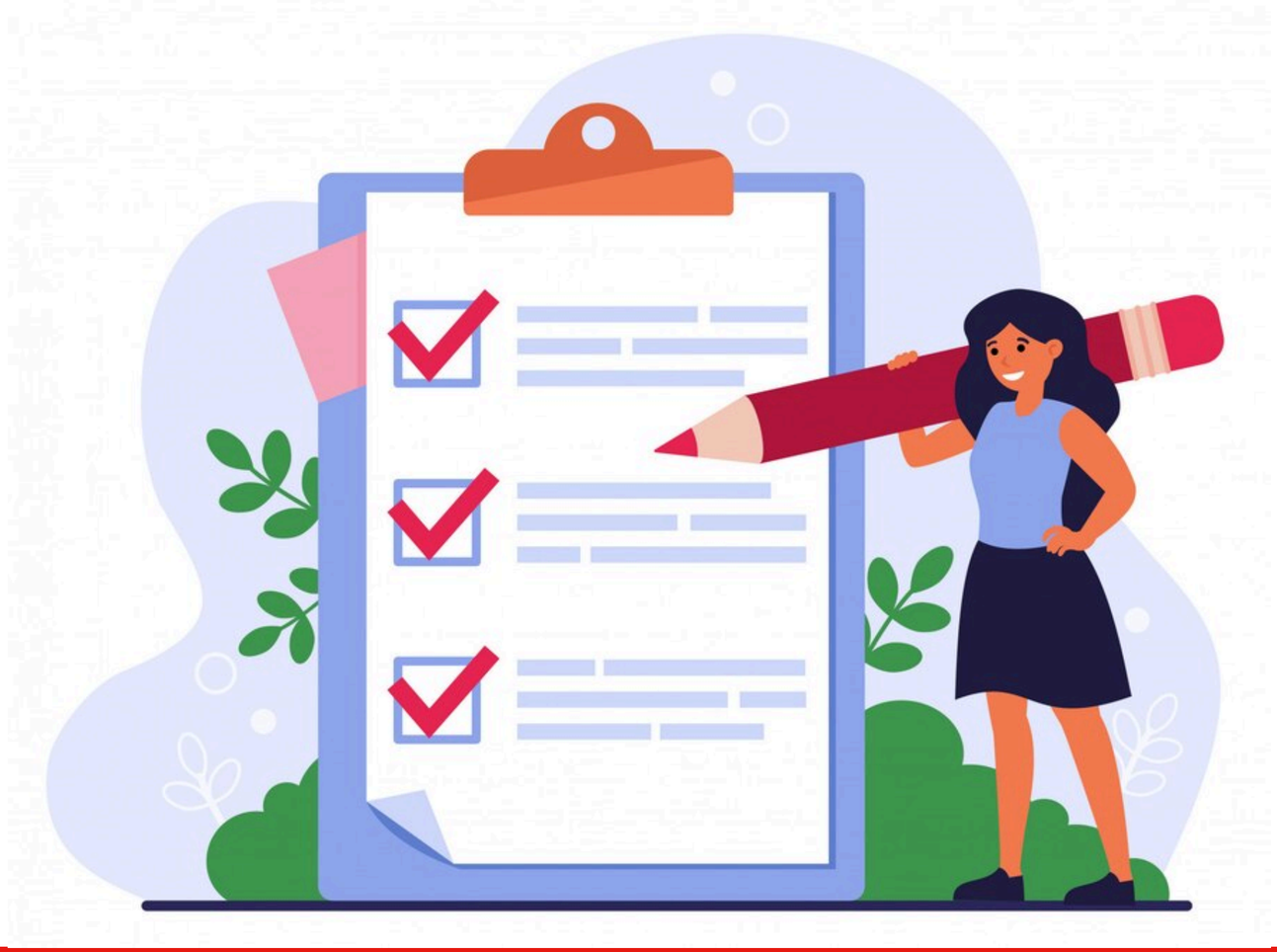


The residual plot indicates the model is generally unbiased, but shows heteroscedasticity, with larger errors for higher predicted values and some outliers present. While predictions form a strong diagonal line, suggesting overall accuracy, refinement is needed for improved predictions at higher demand values.

FEATURES IMPORTANCE



- **Hour:** **Most significant feature; higher values (rush hours) lead to increased bike rentals, while lower values indicate reduced rentals.**
- **Year and Season:** **Higher rentals in specific seasons and later years suggest upward trends in usage.**
- **Weekend Days:** **Mixed effects; higher activity on Saturdays and Sundays due to increased leisure usage.**
- **Weather Features:** **Temperature and humidity strongly correlate with rentals; optimal weather boosts demand, while extreme conditions reduce it.**



CONCLUSION

MODEL PERFORMANCE

XGBoost emerged as the most effective model, with the result after tuning:

- **RMSE:** 39.95 (≈ 4% error at peak demand).
- **MAE:** 24.27 (average deviation of 24 rentals).
- **MAPE:** 24.2% (According to Lewis (1982), the model is classified as “reasonable forecast” accuracy).

FEATURES IMPORTANCE

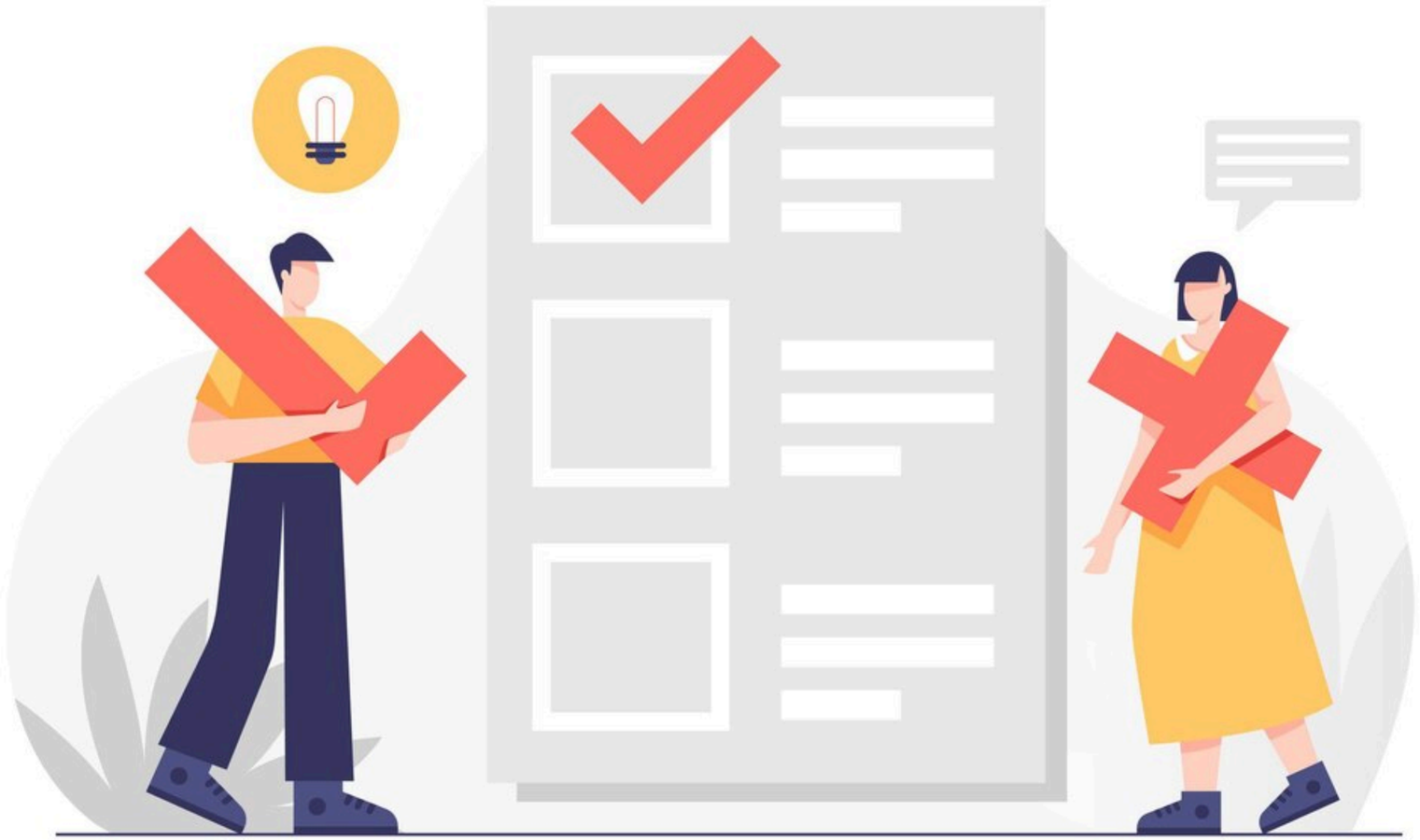
The **hour** feature is crucial, with later hours linked to higher bike rentals.

IMPLICATIONS OF ERRORS

- **Operational Impact:** Underestimating demand may lead to insufficient bikes; overestimating can result in excess costs.
- **Economic Impact:** Assuming an average rental price of $2, a 40-rental underestimation could mean missed revenue of up to $80/hour.

MODEL LIMITATIONS

- **Data Dependency:** Performance relies on quality input features; missing data can affect predictions.
- **Temporal Dynamics:** May not capture sudden changes in user behavior from external factors.
- **Feature Limitations:** Missing socio-economic variables and specific events may impact prediction accuracy.



RECOMMENDATIONS

1. **Optimize Resource Allocation:** Ensure additional bikes are available during the late afternoon and evening hours based on demand predictions.
2. **Enhance Station Operations:** Maintain consistent bike availability year-round and consider summer promotions to attract casual users.
3. **Weather Considerations:** Develop marketing campaigns for clear days and partner with local businesses for discounts to encourage rentals.
4. **Improve Weekend Promotional Efforts:** Implement strategies to attract more weekend users through promotional events and family activities.
5. **Incorporate Additional Features:** Evaluate the impact of factors like special events and weekdays/weekends on predictive performance.
6. **Monitor and Adapt:** Regularly evaluate model performance and adjust strategies based on real-time data; conduct A/B testing on promotions for optimal results.

By focusing on these strategies, stakeholders can enhance bike-sharing operations and improve service efficiency year-round.

THANK YOU