

CSP7040 : ML Ops

Lab Report



॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

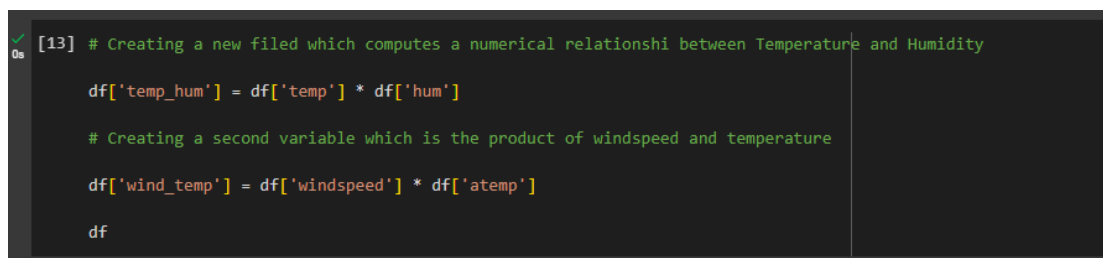
Name: **PINAQ SHARMA**
Roll Number: **M23EET007**
Program: **M.Tech SIoT**

Lab-1

1. Create at least two new interaction features between numerical variables (e.g., temp * hum). Justify your choice of features and explain how they might improve the model's predictive performance.

The Two new features created are "work_active" and "holiday_clear",

- work_active : This field is 1 if the 'workingday' column is 1 and 'hr' is between 9 and 17. This is to check if the duration in which the people are working in their jobs have an impact on the Bike Sharing statistics
- holiday_clear : This field is 1 if the 'holiday' and 'weathersit' are 1. This is to establish a co-relation between the day being an holiday and the weather being clear and assessing its impact on the Bike Sharing.



```
[13] # Creating a new filed which computes a numerical relationshi between Temperature and Humidity

df['temp_hum'] = df['temp'] * df['hum']

# Creating a second variable which is the product of windspeed and temperature

df['wind_temp'] = df['windspeed'] * df['atemp']

df
```

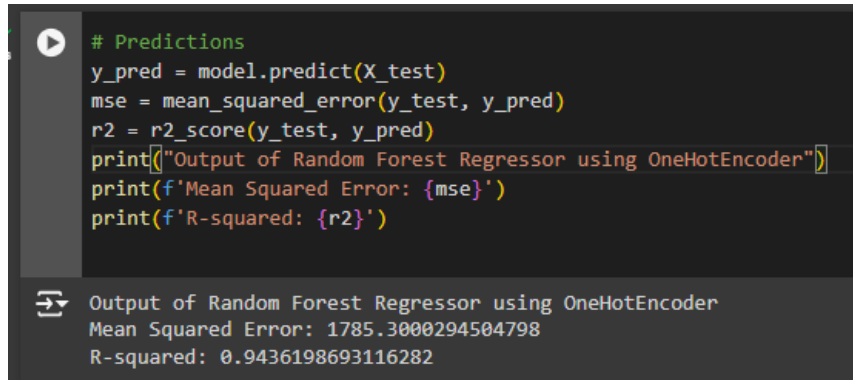
Figure 1: Adding additional iterations

2. Replace the OneHotEncoder with TargetEncoder for categorical variables. Evaluate how this change impacts the model's performance compared to one-hot encoding.

```
1 # Numerical features
2 numerical_features = ['temp', 'atemp', 'hum', 'windspeed', 'temp_hum', 'wind_temp',
3 ↪ ]
4 numerical_pipeline = Pipeline([
5     ('imputer', SimpleImputer(strategy='mean')), # Impute missing values with mean
6     ('scaler', MinMaxScaler()) # Normalize using MinMaxScaler
7 ])
8 # Transforming above
9 X[numerical_features] = numerical_pipeline.fit_transform(X[numerical_features])
10 # Categorical features
11 categorical_features = ['season', 'weathersit']
12 categorical_pipeline = Pipeline([
13     ('imputer', SimpleImputer(strategy='most_frequent')),
14     ('target_encoder', TargetEncoder())
15 ])
16 # Transforming above
17 X_encoded = categorical_pipeline.fit_transform(X[categorical_features], y)
18 # Converting it to a dataframe
19 X_encoded = pd.DataFrame(X_encoded,
20     columns=categorical_pipeline.named_steps['target_encoder'].get_feature_names_out(
21     ↪ categorical_features))
22 # Encoded categorical features + Numerical features
23 X = pd.concat([X.drop(columns=categorical_features), X_encoded], axis=1)
24 X.columns = X.columns.astype(str)
```

Comparison based on the use of OneHotEncoder and TargetEncoder

- Outputs from OneHotEncoder using Random Forest Regressor:-

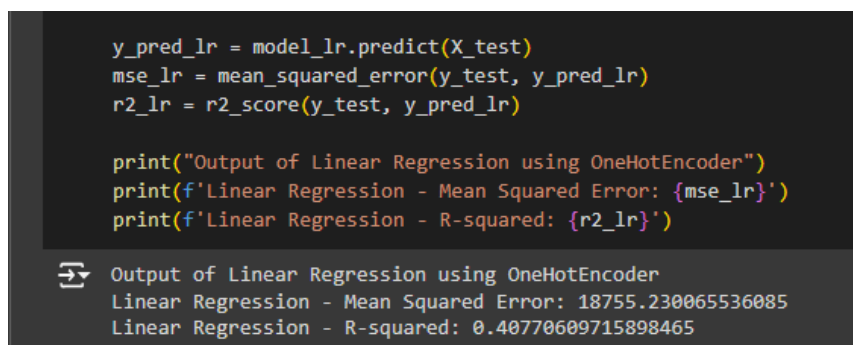


```
# Predictions
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(["Output of Random Forest Regressor using OneHotEncoder"])
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Output of Random Forest Regressor using OneHotEncoder
Mean Squared Error: 1785.3000294504798
R-squared: 0.9436198693116282

Figure 2: Output of Random Forest Regressor

- Outputs from OneHotEncoder using Linear Regression:-



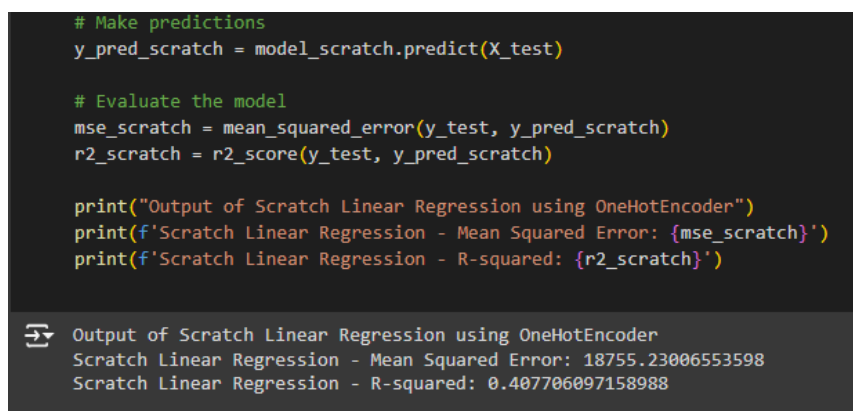
```
y_pred_lr = model_lr.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

print("Output of Linear Regression using OneHotEncoder")
print(f'Linear Regression - Mean Squared Error: {mse_lr}')
print(f'Linear Regression - R-squared: {r2_lr}')
```

Output of Linear Regression using OneHotEncoder
Linear Regression - Mean Squared Error: 18755.230065536085
Linear Regression - R-squared: 0.40770609715898465

Figure 3: Output of Linear Regression

- Outputs from OneHotEncoder using Linear Regression written from scratch:-



```
# Make predictions
y_pred_scratch = model_scratch.predict(X_test)

# Evaluate the model
mse_scratch = mean_squared_error(y_test, y_pred_scratch)
r2_scratch = r2_score(y_test, y_pred_scratch)

print("Output of Scratch Linear Regression using OneHotEncoder")
print(f'Scratch Linear Regression - Mean Squared Error: {mse_scratch}')
print(f'Scratch Linear Regression - R-squared: {r2_scratch}')
```

Output of Scratch Linear Regression using OneHotEncoder
Scratch Linear Regression - Mean Squared Error: 18755.23006553598
Scratch Linear Regression - R-squared: 0.407706097158988

Figure 4: Output of Linear Regression written from scratch

- Outputs from TargetEncoder using Random Forest Regressor:-

```

✓ [36] # Predictions
0s y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print("Output of Random Forest Regressor using TargetEncoder")
    print(f'Mean Squared Error: {mse}')
    print(f'R-squared: {r2}')

```


 Output of Random Forest Regressor using TargetEncoder
 Mean Squared Error: 1785.3000294504798
 R-squared: 0.9436198693116282

Figure 5: Output of Random Forest Regressor

- Outputs from TargetEncoder using Linear Regression:-

```

y_pred_lr = model_lr.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

print("Output of Linear Regression using TargetEncoder")
print(f'Linear Regression - Mean Squared Error: {mse_lr}')
print(f'Linear Regression - R-squared: {r2_lr}')

```


 Output of Linear Regression using TargetEncoder
 Linear Regression - Mean Squared Error: 19139.70080092009
 Linear Regression - R-squared: 0.3955644347217322

Figure 6: Output of Linear Regression

- Outputs from TargetEncoder using Linear Regression written from scratch:-

```

print("Output of Scratch Linear Regression using TargetEncoder")
print(f'Scratch Linear Regression - Mean Squared Error: {mse_scratch}')
print(f'Scratch Linear Regression - R-squared: {r2_scratch}')

```


 Output of Scratch Linear Regression using TargetEncoder
 Scratch Linear Regression - Mean Squared Error: 19139.700800919163
 Scratch Linear Regression - R-squared: 0.3955644347217614

Figure 7: Output of Linear Regression written from scratch

3. Training using LinearRegressor:

1. Using the builtin package

```

1  # Using the Linear Regression Package
2
3  from sklearn.linear_model import LinearRegression
4  from sklearn.metrics import mean_squared_error, r2_score
5
6  model_lr = LinearRegression()
7  model_lr.fit(X_train, y_train)
8
9  y_pred_lr = model_lr.predict(X_test)
10 mse_lr = mean_squared_error(y_test, y_pred_lr)
11 r2_lr = r2_score(y_test, y_pred_lr)
12
13 print("Output of Linear Regression using TargetEncoder")
14 print(f'Linear Regression - Mean Squared Error: {mse_lr}')
15 print(f'Linear Regression - R-squared: {r2_lr}')
```

2. Writing the LinearRegressor from scratch

```

1  import numpy as np
2  import pandas as pd
3  from sklearn.metrics import mean_squared_error, r2_score
4
5  class LinearRegressionScratch:
6      def __init__(self):
7          self.weights = None
8          self.bias = None
9          self.coef_ = None
10
11     def fit(self, X, y):
12         X = np.array(X)
13         y = np.array(y)
14         # Adding a column of ones for the bias term
15         X_b = np.c_[np.ones((X.shape[0], 1)), X]
16         # Calculating weights using the normal equation
17         self.weights = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
18         self.bias = self.weights[0] # First element is the bias (intercept)
19         self.coef_ = self.weights[1:] # Remaining elements are the feature
20                                     ↪ coefficients
21
22     def predict(self, X):
23         X = np.array(X)
24         return X.dot(self.coef_) + self.bias
25
26     def get_feature_importances(self):
27         # Returns the coefficients as a measure of feature importance.
28         return self.coef_
29
30 # Initialize and train the scratch model
31 model_scratch = LinearRegressionScratch()
32 model_scratch.fit(X_train, y_train)
33
34 # Make predictions
35 y_pred_scratch = model_scratch.predict(X_test)
36
37 # Evaluate the model
38 mse_scratch = mean_squared_error(y_test, y_pred_scratch)
39 r2_scratch = r2_score(y_test, y_pred_scratch)
```

```

40 print("Output of Scratch Linear Regression using TargetEncoder")
41 print(f'Scratch Linear Regression - Mean Squared Error: {mse_scratch}')
42 print(f'Scratch Linear Regression - R-squared: {r2_scratch}')

```

Performance Analysis

	Mean Squared Error	R-squared
Random Forest Regressor (OneHotEncoder)	1785.3000294504798	0.9436198693116282
Linear Regression (OneHotEncoder)	18755.230065536085	0.40770609715898465
Scratch Linear Regression (OneHotEncoder)	18755.23006553598	0.407706097158988
Random Forest Regressor (TargetEncoder)	1785.3000294504798	0.9436198693116282
Linear Regression (TargetEncoder)	19139.70080092009	0.3955644347217322
Scratch Linear Regression (TargetEncoder)	19139.700800919163	0.3955644347217614

Table 1: Performance analysis

3. MLflow Pipelines for each training

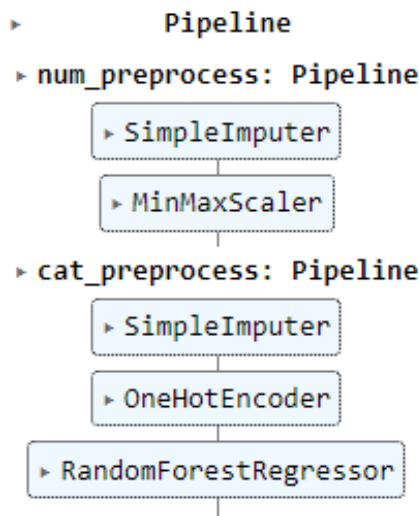


Figure 8: Pipeline for Random Forest using OneHotEncoder

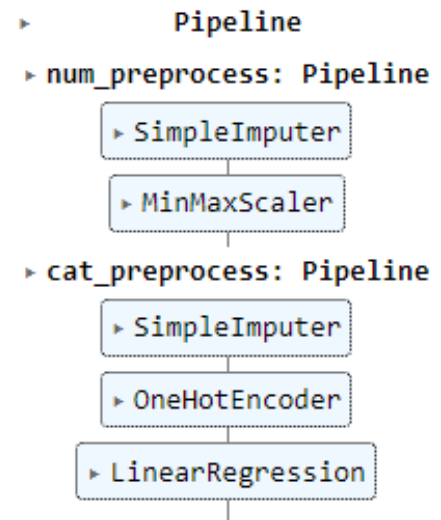


Figure 9: Pipeline for Linear Regressor using OneHotEncoder

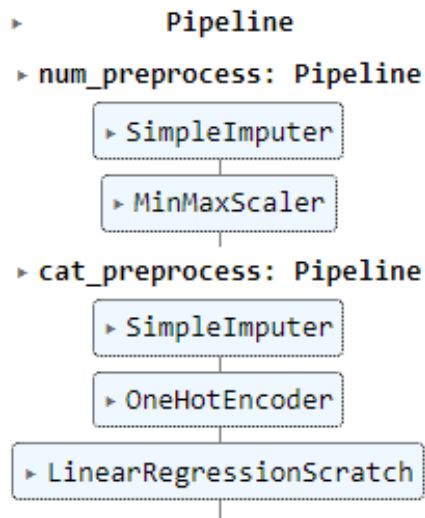


Figure 10: Pipeline for Linear Regressor written from scratch using OneHotEncoder

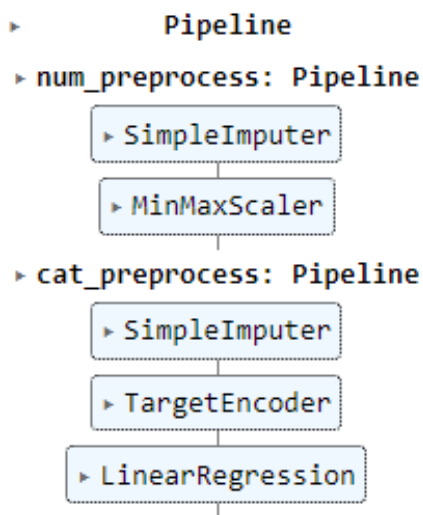


Figure 12: Pipeline for Linear Regressor using TargetEncoder

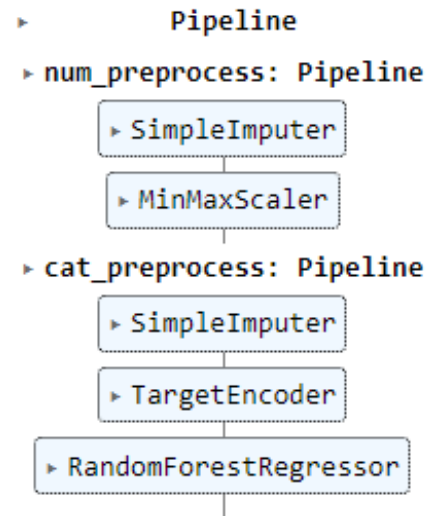


Figure 11: Pipeline for Random Forest using TargetEncoder

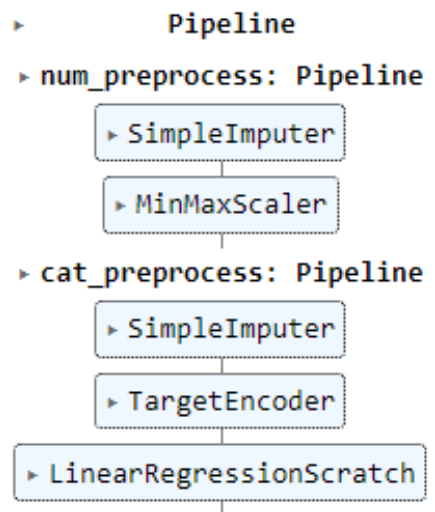


Figure 13: Pipeline for Linear Regressor written from scratch using TargetEncoder

4. Analysis of the importance of various parameters in the dataset

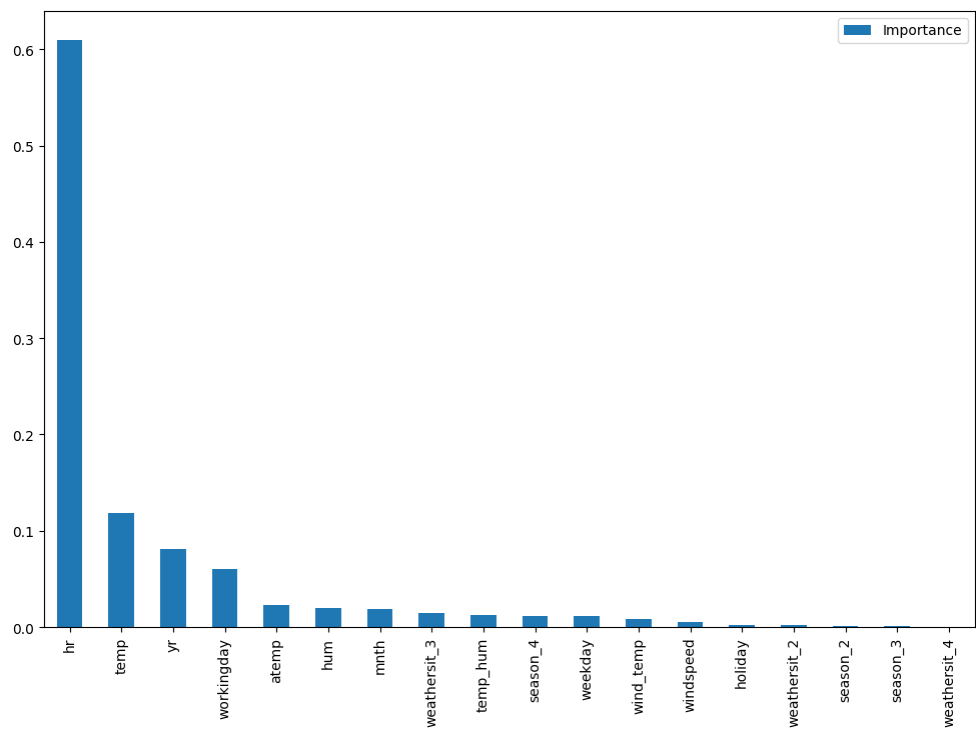


Figure 14: Importance analysis of Random Forest Regressor using OneHotEncoder

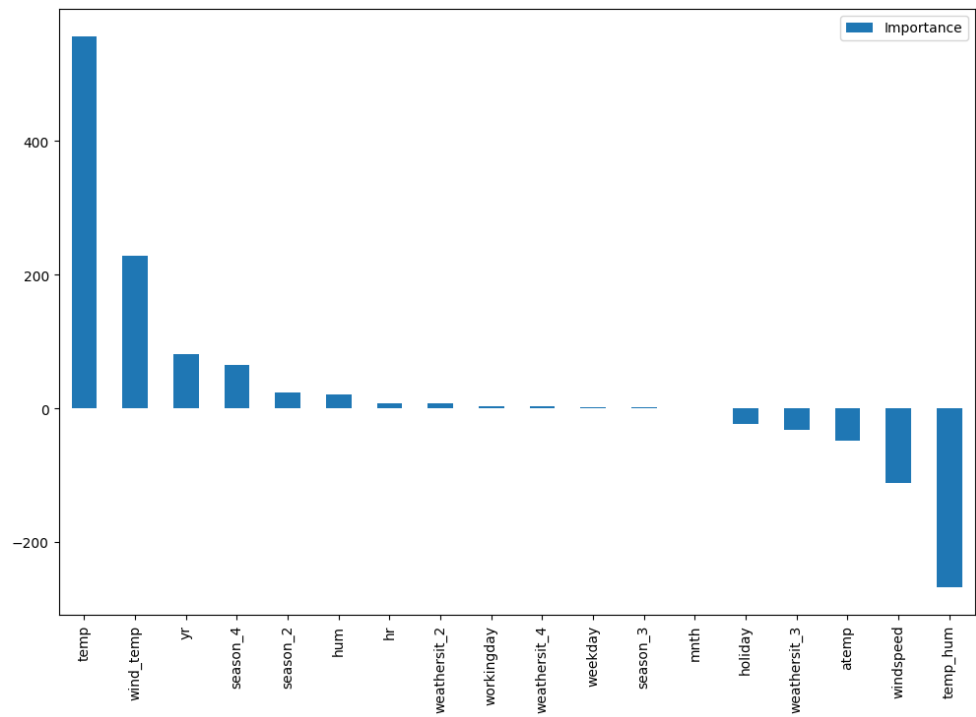


Figure 15: Importance analysis of Linear Regressor using OneHotEncoder

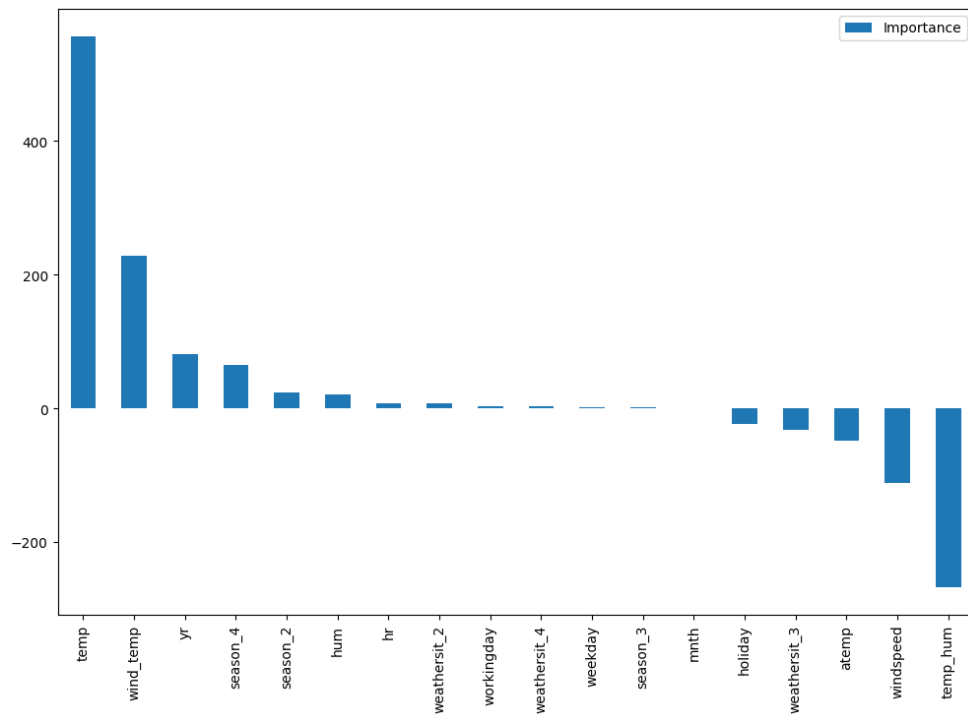


Figure 16: Importance analysis of Linear Regressor written from scratch using OneHotEncoder

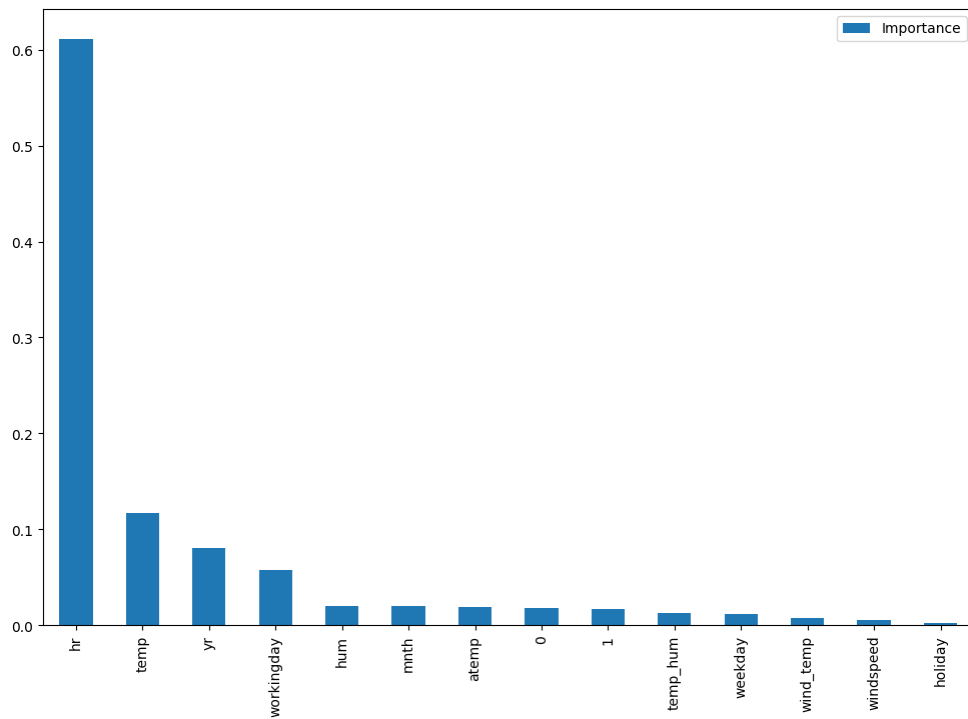


Figure 17: Importance analysis of Random Forest Regressor using TargetEncoder

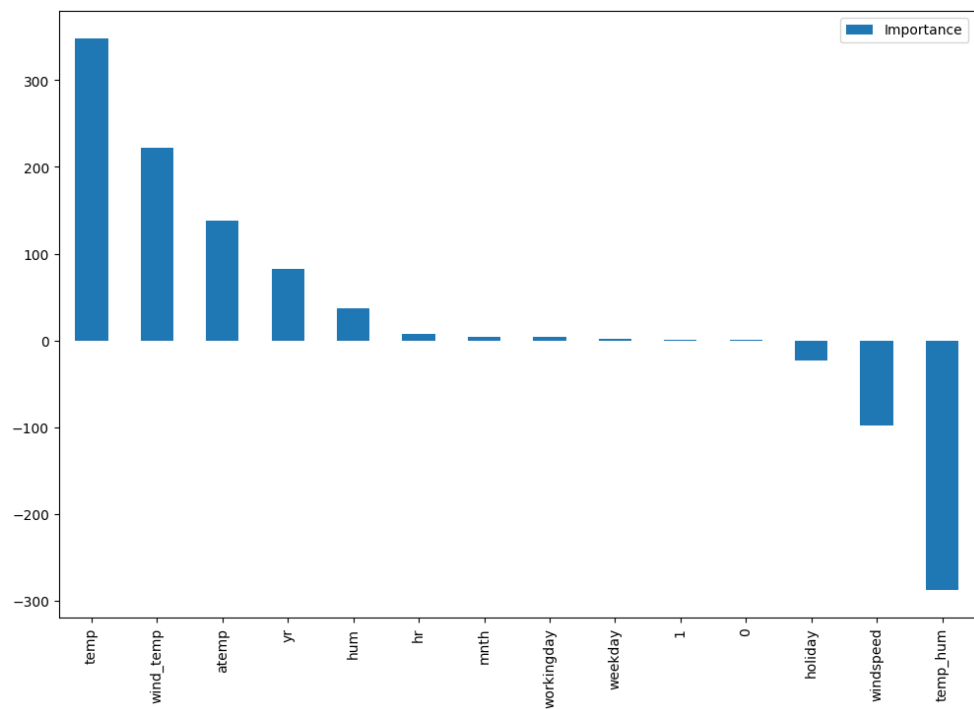


Figure 18: Importance analysis of Linear Regressor using TargetEncoder

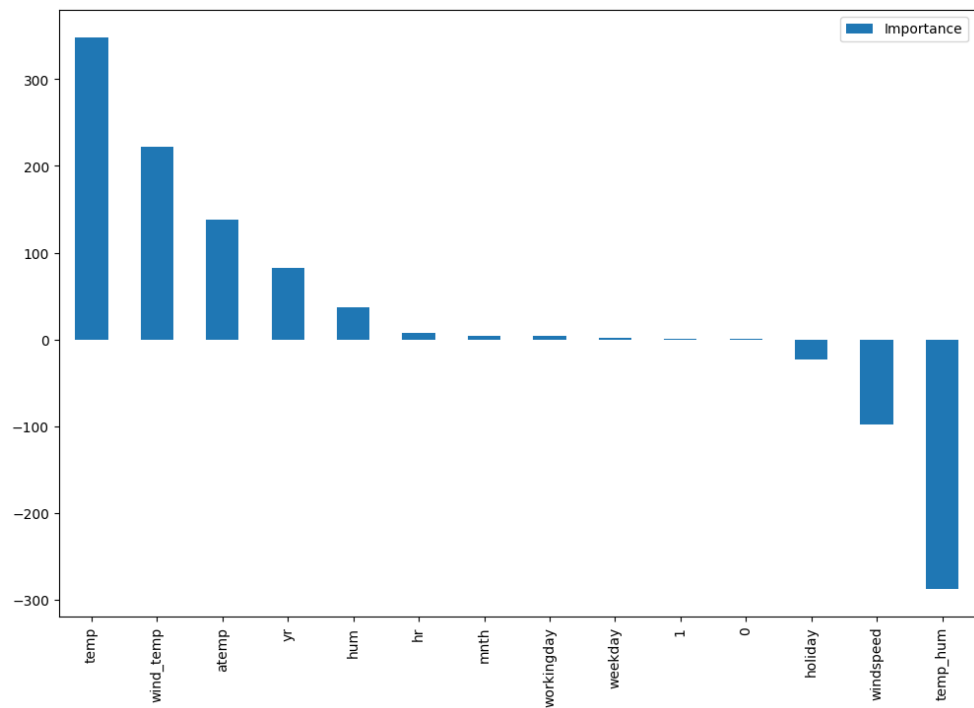


Figure 19: Importance analysis of Linear Regressor written from scratch using TargetEncoder