

MLOps - ASSIGNMENT -2

Name: Gubbala Mohith Nukesh

Roll No:M24CSA037

Linear Regression Model

STEP 1:

Data is taken from given zip file

```
!wget https://archive.ics.uci.edu/static/public/275/bike+sharing+dataset.zip
```

```
!unzip bike+sharing+dataset.zip
```

STEP 2:

For given dataset correlation between them is obtained

	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered
season	1.000000	-0.010742	0.830386	-0.006117	-0.009585	-0.002335	0.013743	-0.014524	0.312025	0.319380	0.150625	-0.149773	0.120206	0.120206
yr	-0.010742	1.000000	-0.010473	-0.003867	0.006692	-0.004485	-0.002196	-0.019157	0.040913	0.039222	-0.083546	-0.008740	0.142779	0.142779
mnth	0.830386	-0.010473	1.000000	-0.005772	0.018430	0.010400	-0.003477	0.005400	0.201691	0.208096	0.164411	-0.135386	0.068457	0.068457
hr	-0.006117	-0.003867	-0.005772	1.000000	0.000479	-0.003498	0.002285	-0.020203	0.137603	0.133750	-0.276498	0.137252	0.301202	0.301202
holiday	-0.009585	0.006692	0.018430	0.000479	1.000000	-0.102088	-0.252471	-0.017036	-0.027340	-0.030973	-0.010588	0.003988	0.031564	0.031564
weekday	-0.002335	-0.004485	0.010400	-0.003498	-0.102088	1.000000	0.035955	0.003311	-0.001795	-0.008821	-0.037158	0.011502	0.032721	0.032721
workingday	0.013743	-0.002196	-0.003477	0.002285	-0.252471	0.035955	1.000000	0.044672	0.055390	0.054667	0.015688	-0.011830	-0.300942	-0.300942
weathersit	-0.014524	-0.019157	0.005400	-0.020203	-0.017036	0.003311	0.044672	1.000000	-0.102640	-0.105563	0.418130	0.026226	-0.152628	-0.152628
temp	0.312025	0.040913	0.201691	0.137603	-0.027340	-0.001795	0.055390	-0.102640	1.000000	0.987672	-0.069881	-0.023125	0.459616	0.459616
atemp	0.319380	0.039222	0.208096	0.133750	-0.030973	-0.008821	0.054667	-0.105563	0.987672	1.000000	-0.051918	-0.062336	0.454080	0.454080
hum	0.150625	-0.083546	0.164411	-0.276498	-0.010588	-0.037158	0.015688	0.418130	-0.069881	-0.051918	1.000000	-0.290105	-0.347028	-0.347028
windspeed	-0.149773	-0.008740	-0.135386	0.137252	0.003988	0.011502	-0.011830	0.026226	-0.023125	-0.062336	-0.290105	1.000000	0.090287	0.090287
casual	0.120206	0.142779	0.068457	0.301202	0.031564	0.032721	-0.300942	-0.152628	0.459616	0.454080	-0.347028	0.090287	1.000000	1.000000

Comparing the above table to create two new interaction features between numerical variables using correlation.

Comparing correlation, it is found that between humidity & windspeed (-0.290105), Temperature & humidity (0.069881) are more related than other relations.

Humidity & Windspeed: -0.290105

Negative Correlation: A correlation of -0.290105 indicates a weak to moderate negative relationship between humidity and windspeed. As humidity increases, windspeed tends to decrease slightly, and vice versa. The negative sign means that when one feature increases, the other tends to decrease.

Temperature & Humidity: 0.069881

Positive Correlation: A correlation of 0.069881 indicates a very weak positive relationship between temperature and humidity. As temperature increases,

humidity increases very slightly, but the relationship is weak, meaning there's almost no meaningful trend between these two features.

Adding these new columns to DataFrame

```
df['wind*hum'] = df['windspeed'] *df["hum"]
```

```
df['temp*hum'] =df['temp'] *df["hum"]
```

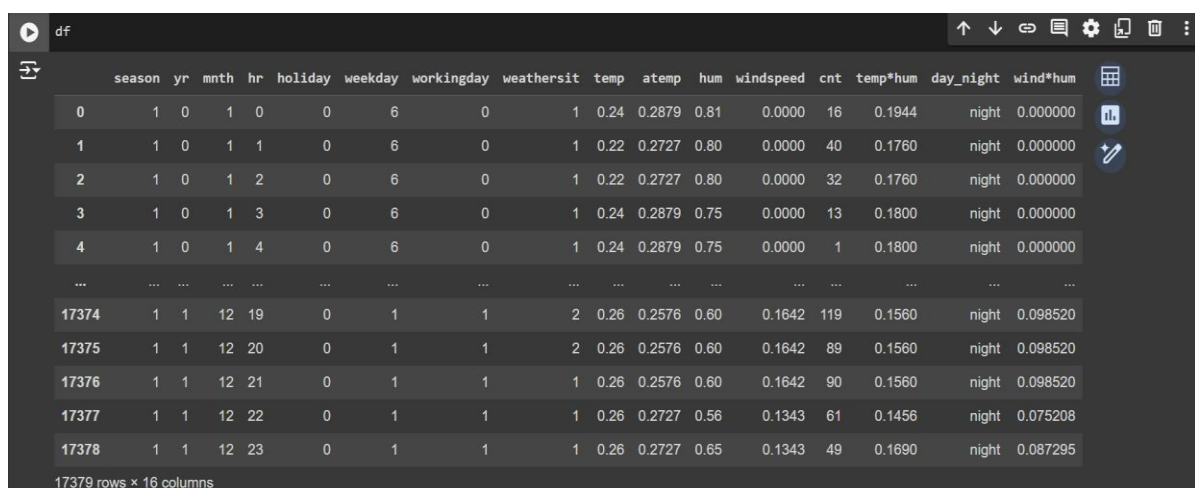
Step 3:

```
# Separating features and target variable
```

```
X = df.drop(columns=['cnt']) # Features
```

```
y = df['cnt'] # Target
```

Printing the X[features] DataFrame



	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	cnt	temp*hum	day_night	wind*hum
0	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0000	16	0.1944	night	0.000000
1	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0000	40	0.1760	night	0.000000
2	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0000	32	0.1760	night	0.000000
3	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0000	13	0.1800	night	0.000000
4	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0000	1	0.1800	night	0.000000
...
17374	1	1	12	19	0	1	1	2	0.26	0.2576	0.60	0.1642	119	0.1560	night	0.098520
17375	1	1	12	20	0	1	1	2	0.26	0.2576	0.60	0.1642	89	0.1560	night	0.098520
17376	1	1	12	21	0	1	1	1	0.26	0.2576	0.60	0.1642	90	0.1560	night	0.098520
17377	1	1	12	22	0	1	1	1	0.26	0.2727	0.56	0.1343	61	0.1456	night	0.075208
17378	1	1	12	23	0	1	1	1	0.26	0.2727	0.65	0.1343	49	0.1690	night	0.087295

STEP 3: IMPUTATION

```
from sklearn.impute import SimpleImputer
```

```
from sklearn.preprocessing import TargetEncoder, MinMaxScaler
```

```
from sklearn.pipeline import Pipeline
```

```
# Numerical features
```

```
numerical_features = ['temp', 'hum', 'windspeed', 'wind*hum', 'temp*hum']
```

```
numerical_pipeline = Pipeline([
```

```
('imputer', SimpleImputer(strategy='mean')), # Impute missing values with mean
```

```
(scaler', MinMaxScaler()) # Normalize using MinMaxScaler
])

# Transforming above
X[numerical_features] = numerical_pipeline.fit_transform(X[numerical_features])
```

The above code imputes missing values using the mean with SimpleImputer, and then normalizes the data to a range between 0 and 1 using MinMaxScaler. This processed data is then applied to the specified numerical features in the DataFrame X.

Printing categorical_features

```
categorical_features = ['season', 'weathersit', 'day_night']
X_features = X[categorical_features]
X_features.head()
```

	season	weathersit	day_night
0	1	1	night
1	1	1	night
2	1	1	night
3	1	1	night
4	1	1	night

For categorical features

Using TargetEncoder

```
import category_encoders as ce

# Categorical features
categorical_features = ['season', 'weathersit', 'day_night']

categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('target_encoder', ce.TargetEncoder()) # Use TargetEncoder from category_encoders
])

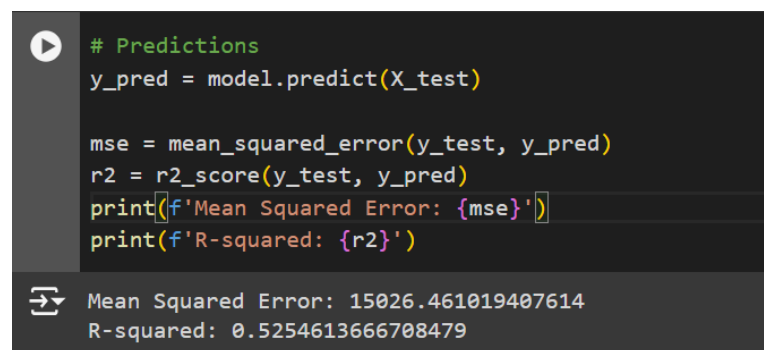
# Transforming above
X_encoded = categorical_pipeline.fit_transform(X[categorical_features], y)
```

```
# Converting it to a dataframe (might not be needed with category_encoders)
X_encoded = pd.DataFrame(X_encoded,
columns=categorical_pipeline.named_steps['target_encoder'].get_feature_names())

# Encoded categorical features + Numerical features
X = pd.concat([X.drop(columns=categorical_features), X_encoded], axis=1)
```

1) Multiple linear regression using package & TargetEncoder

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
model = LinearRegression()
model.fit(X_train, y_train)
```



```
# Predictions
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Mean Squared Error: 15026.461019407614
R-squared: 0.5254613666708479

2) Multiple linear regression from scratch & TargetEncoder

```
import numpy as np
def fit(X, y):
    X = np.c_[np.ones(X.shape[0]), X]
    coefficients = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
    return coefficients
def predict(X, coefficients):
    X = np.c_[np.ones(X.shape[0]), X]
    return X.dot(coefficients)

def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

def r2_score(y_true, y_pred):
    total_variance = np.sum((y_true - np.mean(y_true)) ** 2)
    residual_variance = np.sum((y_true - y_pred) ** 2)
    return 1 - (residual_variance / total_variance)

coefficients = fit(X_train, y_train)

y_pred = predict(X_test, coefficients)
```

```
print("Multiple linear regression from scratch")
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared Score:", r2_score(y_test, y_pred))
```

```
Multiple linear regression from scratch
Mean Squared Error: 15026.461019406946
R-squared Score: 0.525461366670869
```

Review: The Mean Squared Error (MSE) and R-squared (R^2) values are nearly identical for both the package-based and scratch implementations of the multiple linear regression model, indicating that the performance of the two approaches is virtually equivalent.

Using OneHotEncoder

```
# Categorical features
from sklearn.preprocessing import OneHotEncoder
categorical_features = ['season', 'weathersit', 'day_night']
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(sparse_output=False, drop='first'))
])

# Transforming above
X_encoded = categorical_pipeline.fit_transform(X[categorical_features])

# Converting it to a dataframe
X_encoded = pd.DataFrame(X_encoded,
    columns=categorical_pipeline.named_steps['onehot'].get_feature_names_out(categorical_features))

# Encoded categorical features + Numerical features
X = pd.concat([X.drop(columns=categorical_features), X_encoded], axis=1)

Step 5:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
```

3) Multiple linear regression using package & OneHotEncoder

```
[32] Suggested code may be subject to a license | itpacbaja/Previsao-Ranking
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
model = LinearRegression()
model.fit(X_train, y_train)

LinearRegression
LinearRegression()

[33] y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Mean Squared Error: 3.57270702659527e-27
R-squared: 1.0

4) Multiple linear regression from scratch & OneHotEncoder

```
def predict(X, coefficients):
    X = np.c_[np.ones(X.shape[0]), X]
    return X.dot(coefficients)

def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

def r2_score(y_true, y_pred):
    total_variance = np.sum((y_true - np.mean(y_true)) ** 2)
    residual_variance = np.sum((y_true - y_pred) ** 2)
    return 1 - (residual_variance / total_variance)

coefficients = fit(X_train, y_train)

y_pred = predict(X_test, coefficients)

print("Multiple linear regression from scratch")
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared Score:", r2_score(y_test, y_pred))
```

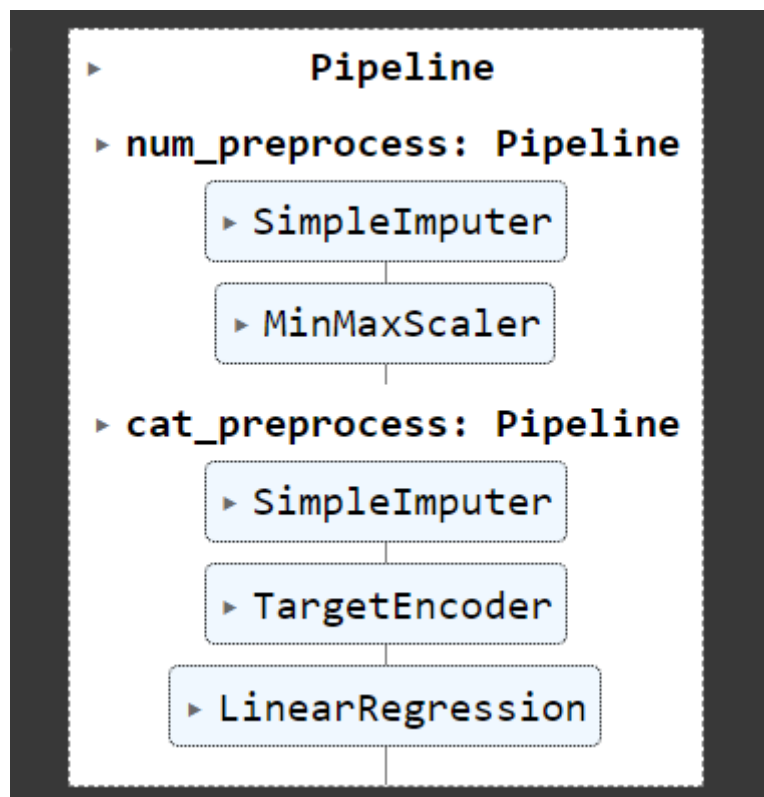
Multiple linear regression from scratch
Mean Squared Error: 5.16517574685394e-21
R-squared Score: 1.0

Review:

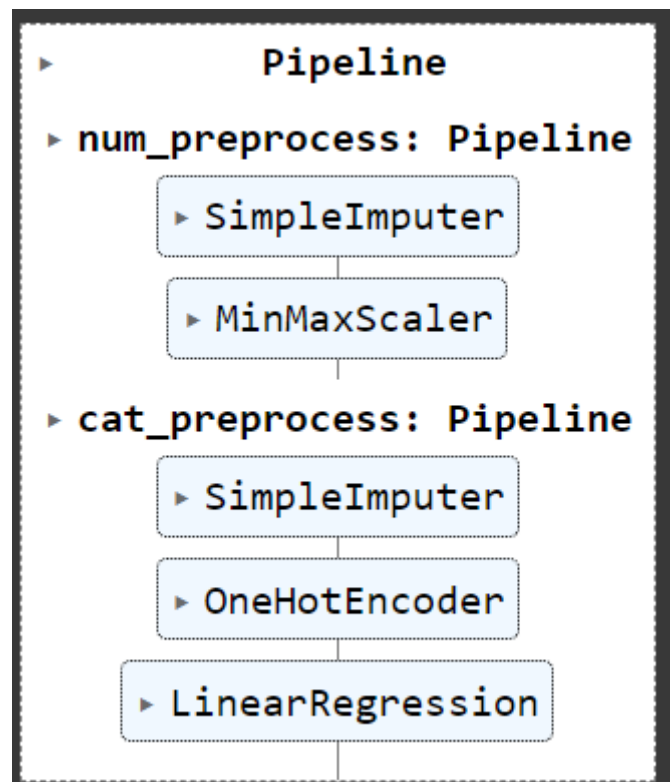
Both the package-based and scratch implementations of multiple linear regression achieve nearly perfect R-squared values and extremely low MSE, indicating that both methods fit the training data exceptionally well. The discrepancies in MSE values suggest numerical precision differences, but overall, the model performance is consistent between approaches.

ML Pipeline

1) Using TargetEncoder & LinearRegression



2) Using OneHotEncoder and LinearRegression



OBSERVATIONS:

TargetEncoder: Encodes categories based on their average target value, which can capture more nuanced relationships but may introduce target leakage and requires careful cross-validation to avoid overfitting.

OneHotEncoder: Creates binary columns for each category, avoiding target leakage and making it easier for models to interpret categorical features, but may result in high-dimensional data with many features.

MODEL PERFORMANCE CONSIDERATION:

TargetEncoder might improve performance with fewer features but risks overfitting if not handled properly.

OneHotEncoder ensures no target leakage and may perform better with algorithms that handle high dimensionality well.

CONCLUSION:

The results show that using `TargetEncoder` gives similar performance for both the built-in and scratch implementations of the linear regression model, with small differences in error and R-squared values. However, using `OneHotEncoder` leads to almost perfect performance metrics for both methods, suggesting that the model fits the training data extremely well. This indicates that one-hot encoding might be better at capturing the data's relationships without any risk of overfitting.