# Bike Sharing — Multiple Linear Regression (Jupyter-ready .python notebook)

# 1. Standard imports

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score, mean\_squared\_error

import statsmodels.api as sm

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

DATA\_PATH = "day.csv" # <-- change if necessary

try:

df = pd.read\_csv(DATA\_PATH)

except Exception as e:

raise SystemExit(f"Could not read {DATA\_PATH}: {e}\nPlease put the dataset file in the notebook folder or update DATA\_PATH.")

print("Data loaded. Shape:", df.shape)

print(df.head())

print('\nColumns:', df.columns.tolist())

print('\nInfo:')

df.info()

cat\_map = {

'season': {1: 'spring', 2: 'summer', 3: 'fall', 4: 'winter'},

'weathersit': {1: 'clear', 2: 'mist', 3: 'light\_snow\_rain', 4: 'heavy\_rain\_snow'},

'mnth': {1:'Jan',2:'Feb',3:'Mar',4:'Apr',5:'May',6:'Jun',7:'Jul',8:'Aug',9:'Sep',10:'Oct',11:'Nov',12:'Dec'},

'weekday': {0:'Sun',1:'Mon',2:'Tue',3:'Wed',4:'Thu',5:'Fri',6:'Sat'}

}

for col, mapping in cat\_map.items():

if col in data.columns:

data[col] = data[col].map(mapping).astype('category')

plt.figure(figsize=(8,4))

plt.hist(data['cnt'], bins=30)

plt.title('Distribution of cnt (target)')

plt.xlabel('cnt')

plt.ylabel('Frequency')

plt.show()

numeric\_cols = data.select\_dtypes(include=[np.number]).columns.tolist()

print('Numeric columns:', numeric\_cols)

plt.figure(figsize=(10,8))

sns.heatmap(data[numeric\_cols].corr(), annot=True, fmt='.2f', cmap='coolwarm')

plt.title('Correlation matrix (numeric features)')

plt.show()

cat\_cols = data.select\_dtypes(include=['category','object']).columns.tolist()

print('Categorical columns to encode:', cat\_cols)

X = data\_enc.drop(columns=['cnt'])

y = data\_enc['cnt']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

print('Train shape:', X\_train.shape, 'Test shape:', X\_test.shape)

vif\_data = pd.DataFrame()

vif\_data['feature'] = X\_train.columns

vif\_data['VIF'] = [variance\_inflation\_factor(X\_train.values, i) for i in range(X\_train.shape[1])]

def backward\_elimination(X, y, sl=0.05):

X\_with\_const = sm.add\_constant(X)

model = sm.OLS(y, X\_with\_const).fit()

pmax = model.pvalues.drop('const').max()

while pmax > sl:

feature\_with\_pmax = model.pvalues.drop('const').idxmax()

print(f"Dropping {feature\_with\_pmax} with p-value {pmax:.4f}")

X = X.drop(columns=[feature\_with\_pmax])

X\_with\_const = sm.add\_constant(X)

model = sm.OLS(y, X\_with\_const).fit()

pmax = model.pvalues.drop('const').max()

return X, model

X\_train\_sel, model\_sel = backward\_elimination(X\_train.copy(), y\_train, sl=0.05)

print('\nFinal model summary after backward elimination:')

print(model\_sel.summary())

lr = LinearRegression()

lr.fit(X\_train\_sel, y\_train)

X\_test\_sel = X\_test[X\_train\_sel.columns]

y\_pred = lr.predict(X\_test\_sel)

print('Test R-squared:', r2\_score(y\_test, y\_pred))

print('Test RMSE:', np.sqrt(mean\_squared\_error(y\_test, y\_pred)))

residuals = y\_test - y\_pred

plt.figure(figsize=(8,4))

plt.scatter(y\_pred, residuals)

plt.axhline(0, color='black', linestyle='--')

plt.xlabel('Predicted cnt')

plt.ylabel('Residuals')

plt.title('Residuals vs Predicted')

plt.show()

import scipy.stats as stats

plt.figure(figsize=(6,6))

stats.probplot(residuals, dist="norm", plot=plt)

plt.title('Q-Q plot of residuals')

plt.show()

print('Shapiro-Wilk test for normality (p-value):')

print(stats.shapiro(residuals))

if (residuals.std() / y\_test.mean()) > 0.5:

print('\nResiduals have high dispersion relative to mean. Trying log-transform of target...')

y\_log = np.log1p(y)

X\_train\_log, X\_test\_log, y\_train\_log, y\_test\_log = train\_test\_split(X, y\_log, test\_size=0.25, random\_state=42)

X\_train\_log\_sm = sm.add\_constant(X\_train\_log)

model\_log = sm.OLS(y\_train\_log, X\_train\_log\_sm).fit()

print(model\_log.summary())

# Fit sklearn on all features (or selected ones from model)