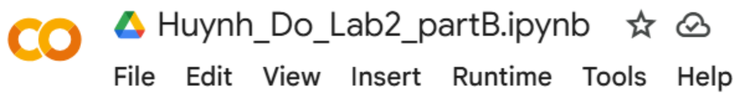
Huynh Do Lab#2 Part B:

Objective: Use Python in Google Colab to download your dataset "bikes.csv", to explore and summarize the Regression



1. **Import libraries**

```python
import pandas as pd
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from google.colab import files
```

The code above imports several Python libraries commonly used in data analysis.

1. import pandas as pd: Brings in Pandas, r go-to library for working with tabular data (like CSV files).
   - Use it to load, clean, manipulate, and analyze datasets.
   - Example: df = pd.read_csv("data.csv")
2. import statsmodels.api as sm :
   - This loads Statsmodels, a library for doing deep-dive statistical analysis.Useful for regression with detailed output: coefficients, p-values, confidence intervals, etc.
   - Commonly used for OLS (Ordinary Least Squares) regression.
3. from sklearn.linear_model import LinearRegression :
   - model.fit(X_train, y_train)
   - Grabs the Linear Regression model from Scikit-learn.
     - Faster, simpler, great for prediction.
4. from sklearn.model_selection import train_test_split
   - Split r dataset into training and testing sets.
   - Ensures 're testing r model on unseen data.
   - X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
5. from google.colab import files
   - Used in Google Colab to upload/download files from your local system within a Colab notebook.
6. from sklearn.metrics import r2_score, mean_squared_error
   - These are tools to evaluate r model's performance.
   - r2_score: How well r model explains the variance (ranges from 0 to 1; higher is better).

- mean_squared_error: Average of the squared differences between predicted and actual values (lower is better).

2. Upload file BIKES.csv

```
# Upload 'NBA.csv' file
uploaded = files.upload()

# Load the dataset
df = pd.read_csv("BIKES.csv")
df.head()
```

The above screen shot is used to upload and load a CSV file named **BIKE.csv** into a Pandas Data Frame in a Google Colab environment.

When upload is done:

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please reru
Saving BIKES.csv to BIKES (8).csv
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 0:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 1:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 2:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 3:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 4:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

3. OLS Regression: count ~ temp

```
[24] # ---------------------
     # OLS Regression: count ~ temp
     # ---------------------
     X_ols1 = sm.add_constant(df[['temp']])  # Add intercept
     ols_model1 = sm.OLS(df['total_rentals'], X_ols1).fit()
     print("OLS Regression: count ~ temp")
     print(ols_model1.summary())
```

1. df[['temp']]
   - Selecting the temperature column from the DataFrame df as r independent variable (X).
   - Double square brackets [[]] keep it as a DataFrame (not a Series), which is what statsmodels expects.
2. sm.add_constant(...)

    - OLS (Ordinary Least Squares) needs an intercept term — a baseline value when all predictors are 0.
    - X_ols1 = sm.add_constant(df[['temp']]) adds a column of 1s to r X matrix so r model can learn the intercept (i.e., y = b0 + b1 * temp).
3. sm.OLS(df['total_rentals'], X_ols1
    - This sets up an OLS regression model, predicting: total_rentals ~ temp where: df['total_rentals'] = dependent variable (what you're trying to predict, aka y)
4. .fit()
   This trains the model using data:
   It computes the best-fit line using least squares and returns an object (ols_model1) that stores all the regression results.
5. print(ols_model1.summary()):
   This prints out a detailed regression summary, including:
    - Coefficients (slope and intercept)
    - Standard errors
6. Summary
    - The result predicts that temp actually has a statistically significant impact — or if it's just hot air.

```
OLS Regression: count ~ temp
                            OLS Regression Results
==============================================================================
Dep. Variable:          total_rentals   R-squared:                       0.156
Model:                            OLS   Adj. R-squared:                  0.156
Method:                 Least Squares   F-statistic:                     2006.
Date:                Sat, 19 Apr 2025   Prob (F-statistic):               0.00
Time:                        00:21:18   Log-Likelihood:                -71125.
No. Observations:               10886   AIC:                         1.423e+05
Df Residuals:                   10884   BIC:                         1.423e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.0462      4.439      1.362      0.173      -2.656      14.748
temp           9.1705      0.205     44.783      0.000       8.769       9.572
==============================================================================
Omnibus:                     1871.687   Durbin-Watson:                   0.369
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3221.966
Skew:                           1.123   Prob(JB):                         0.00
Kurtosis:                       4.434   Cond. No.                         60.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```
✓ 0s completed at 5:21 PM

The above figure is the result of running the OLS Regression python code

4. Sklearn Linear Regression: count ~ temp

```python
# --------------------
# Sklearn Linear Regression: count ~ temp
# --------------------
X = df[['temp']]
y = df['total_rentals']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
```

1. train_test_split(...)
   - X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
   - Splits data:80% for training and 20% for testing
   - random_state=42 ensures reproducibility (so get the same split every time)
2. model = LinearRegression()\
3. model.fit(X_train, y_train)
   - Trains the model using the training data.
   - Learns the best-fit line: total_rentals ≈ b0 + b1 * temp
4. y_pred = model.predict(X_test)
   Predicts rental counts using the test set temperatures.
   Gives a list of predicted values based on the learned model.
5. r2 = r2_score(y_test, y_pred)
   - Calculates R-squared, which tells how well r model explains the variability in the data.
   - Ranges from 0 to 1:
     - 1 = perfect prediction
     - 0 = no better than guessing the mean
6. mse = mean_squared_error(y_test, y_pred)
   MSE = average of squared prediction errors
   Lower = better
   Sensitive to outliers due to squaring
7. rmse = mse ** 0.5
   RMSE = Root Mean Squared Error

```
[17] print("\nSklearn Linear Regression: count ~ temp")
     print(f"R² Score: {r2:.3f}")
     print(f"RMSE: {rmse:.2f}")
```

```
Sklearn Linear Regression: count ~ temp
R² Score: 0.169
RMSE: 165.59
```

The above figure showed the final results

Quick summary:

1. Temperature **does have an effect** on rentals.
2. Warmer days = more rentals. The relationship is statistically significant.