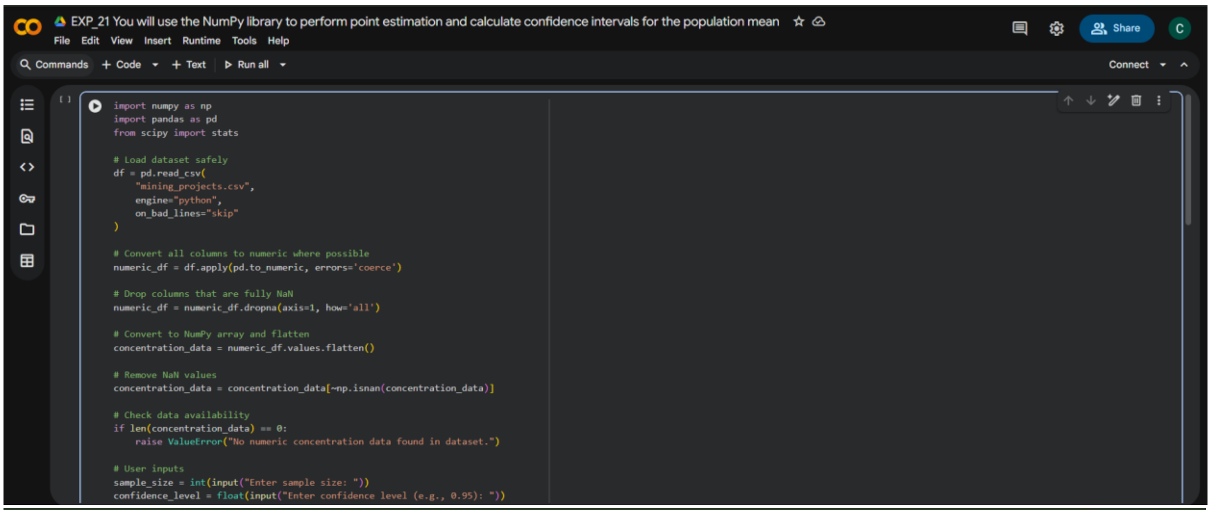
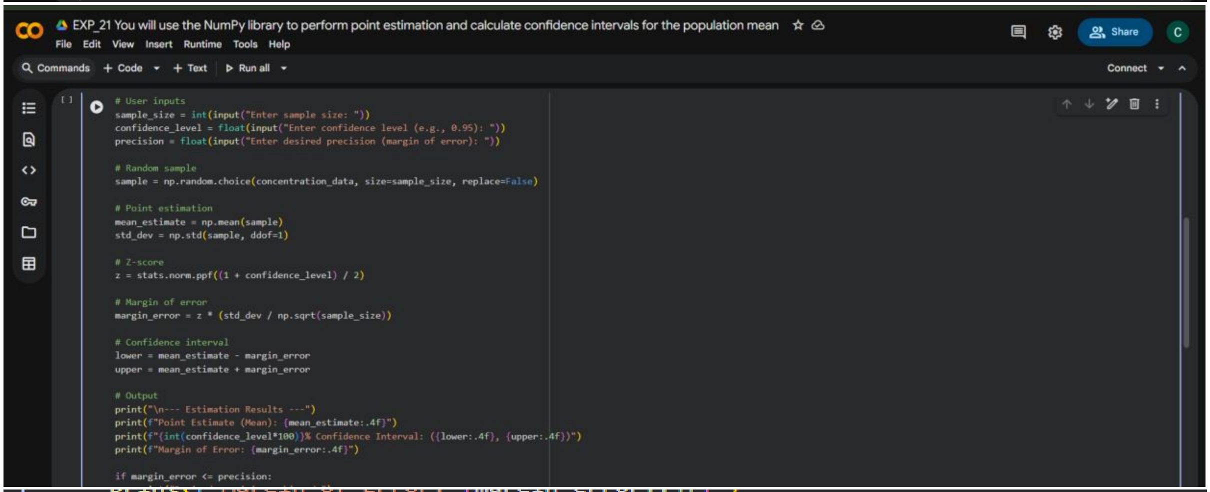
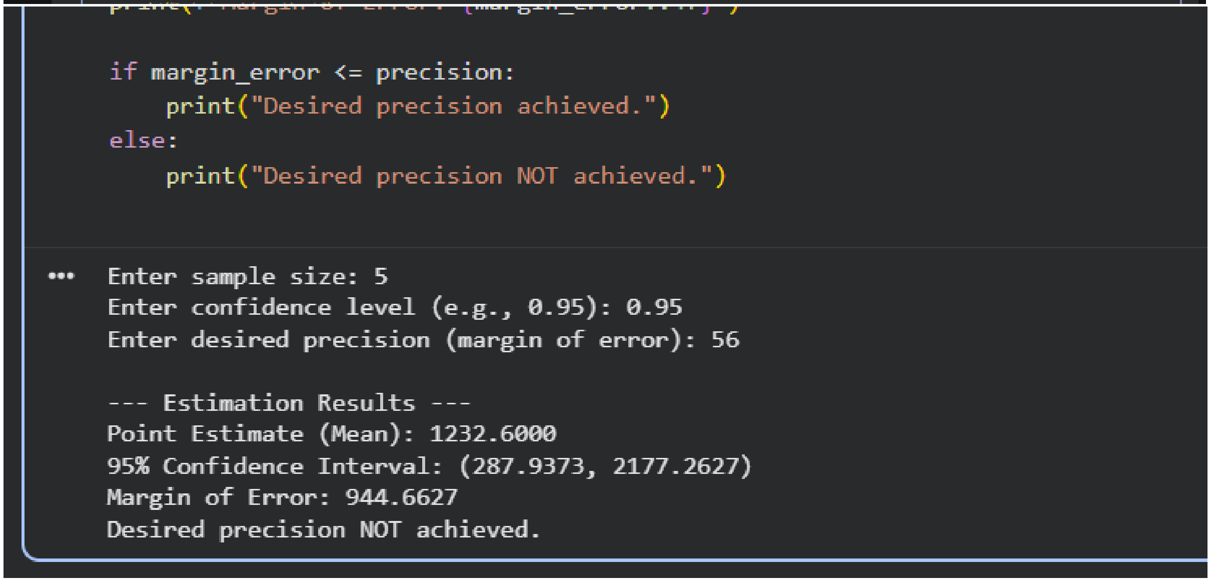
EXP_21


EXP_21 You will use the NumPy library to perform point estimation and calculate confidence intervals for the population mean
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Connect

```
import numpy as np
import pandas as pd
from scipy import stats

# Load dataset safely
df = pd.read_csv(
    "mining_projects.csv",
    engine="python",
    on_bad_lines="skip"
)

# Convert all columns to numeric where possible
numeric_df = df.apply(pd.to_numeric, errors='coerce')

# Drop columns that are fully NaN
numeric_df = numeric_df.dropna(axis=1, how='all')

# Convert to NumPy array and flatten
concentration_data = numeric_df.values.flatten()

# Remove NaN values
concentration_data = concentration_data[~np.isnan(concentration_data)]

# Check data availability
if len(concentration_data) == 0:
    raise ValueError("No numeric concentration data found in dataset.")

# User inputs
sample_size = int(input("Enter sample size: "))
confidence_level = float(input("Enter confidence level (e.g., 0.95): "))
```


EXP_21 You will use the NumPy library to perform point estimation and calculate confidence intervals for the population mean
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Connect

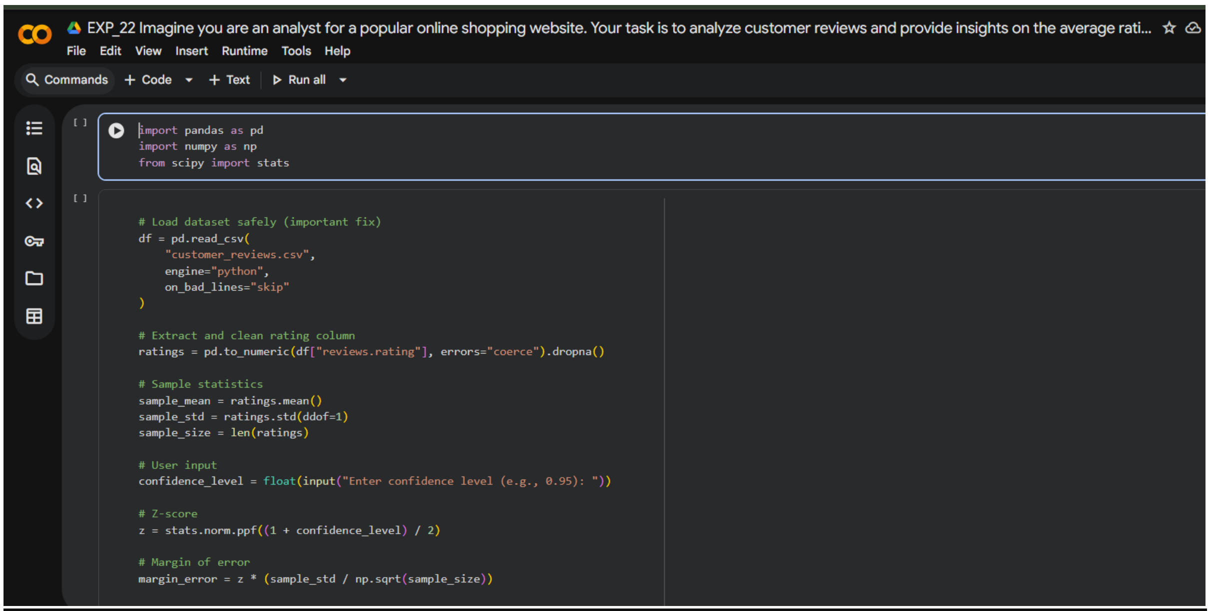
```
# User inputs
sample_size = int(input("Enter sample size: "))
confidence_level = float(input("Enter confidence level (e.g., 0.95): "))
precision = float(input("Enter desired precision (margin of error): "))

# Random sample
sample = np.random.choice(concentration_data, size=sample_size, replace=False)

# Point estimation
mean_estimate = np.mean(sample)
std_dev = np.std(sample, ddof=1)

# Z-score
z = stats.norm.ppf((1 + confidence_level) / 2)

# Margin of error
margin_error = z * (std_dev / np.sqrt(sample_size))

# Confidence interval
lower = mean_estimate - margin_error
upper = mean_estimate + margin_error

# Output
print("\n--- Estimation Results ---")
print(f"Point Estimate (Mean): {mean_estimate:.4f}")
print(f"{int(confidence_level*100)}% Confidence Interval: ({lower:.4f}, {upper:.4f})")
print(f"Margin of Error: {margin_error:.4f}")

if margin_error <= precision:
```



```
if margin_error <= precision:
    print("Desired precision achieved.")
else:
    print("Desired precision NOT achieved.")
```

```
Enter sample size: 5
Enter confidence level (e.g., 0.95): 0.95
Enter desired precision (margin of error): 56

--- Estimation Results ---
Point Estimate (Mean): 1232.6000
95% Confidence Interval: (287.9373, 2177.2627)
Margin of Error: 944.6627
Desired precision NOT achieved.
```

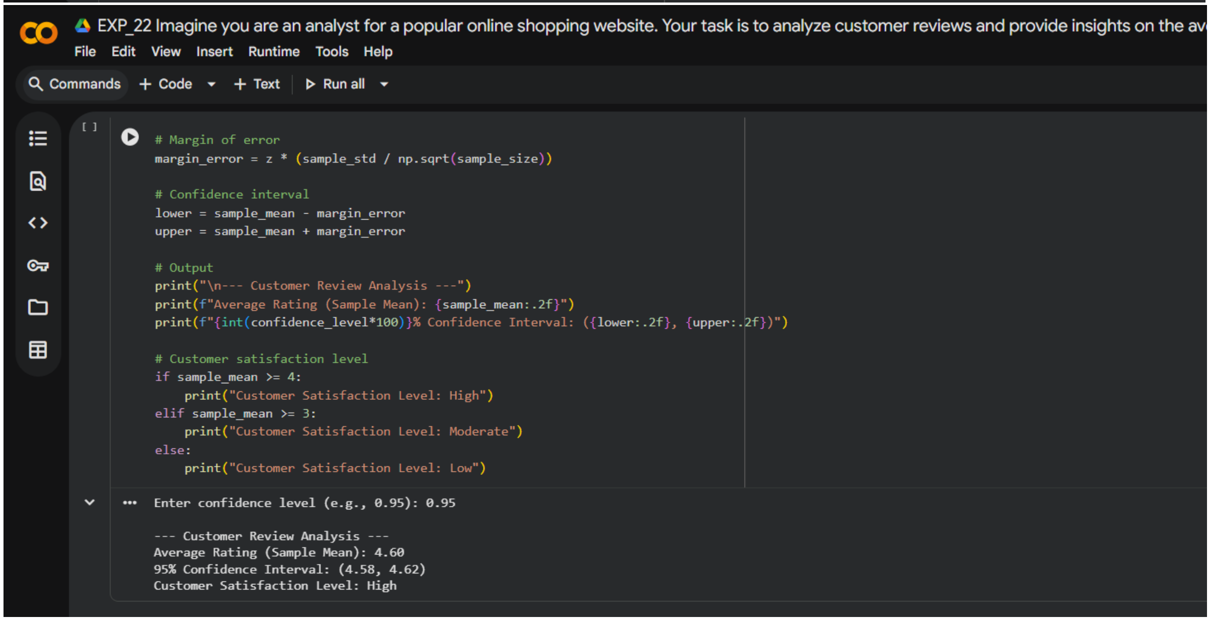
EXP_22



EXP_22 Imagine you are an analyst for a popular online shopping website. Your task is to analyze customer reviews and provide insights on the average rati...
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```python
import pandas as pd
import numpy as np
from scipy import stats
```

```python
# Load dataset safely (important fix)
df = pd.read_csv(
    "customer_reviews.csv",
    engine="python",
    on_bad_lines="skip"
)

# Extract and clean rating column
ratings = pd.to_numeric(df["reviews.rating"], errors="coerce").dropna()

# Sample statistics
sample_mean = ratings.mean()
sample_std = ratings.std(ddof=1)
sample_size = len(ratings)

# User input
confidence_level = float(input("Enter confidence level (e.g., 0.95): "))

# Z-score
z = stats.norm.ppf((1 + confidence_level) / 2)

# Margin of error
margin_error = z * (sample_std / np.sqrt(sample_size))
```

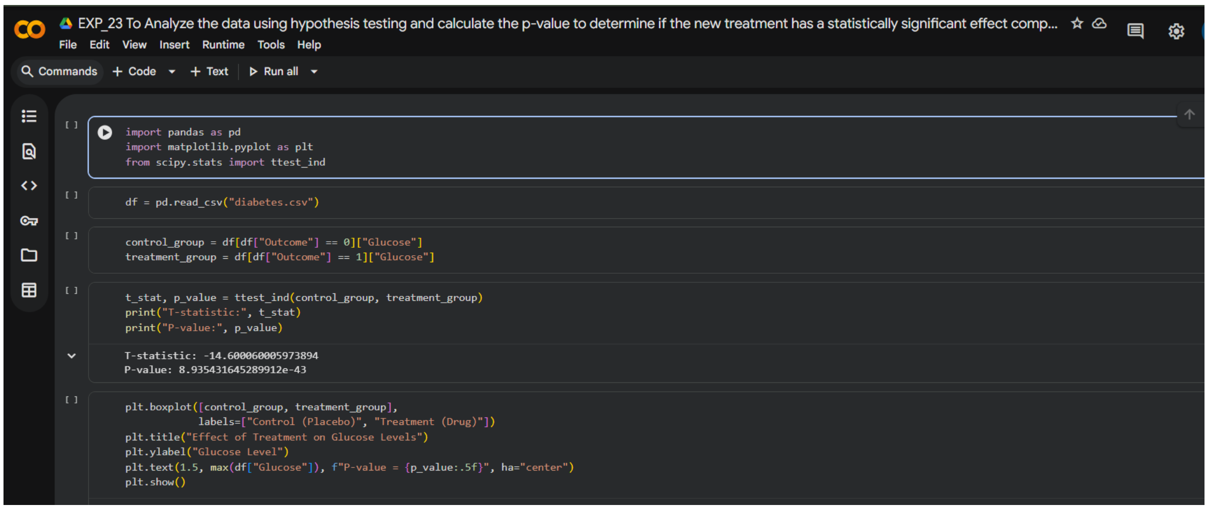


EXP_22 Imagine you are an analyst for a popular online shopping website. Your task is to analyze customer reviews and provide insights on the av
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```python
# Margin of error
margin_error = z * (sample_std / np.sqrt(sample_size))

# Confidence interval
lower = sample_mean - margin_error
upper = sample_mean + margin_error

# Output
print("\n--- Customer Review Analysis ---")
print(f"Average Rating (Sample Mean): {sample_mean:.2f}")
print(f"{int(confidence_level*100)}% Confidence Interval: ({lower:.2f}, {upper:.2f})")

# Customer satisfaction level
if sample_mean >= 4:
    print("Customer Satisfaction Level: High")
elif sample_mean >= 3:
    print("Customer Satisfaction Level: Moderate")
else:
    print("Customer Satisfaction Level: Low")
```

```
Enter confidence level (e.g., 0.95): 0.95

--- Customer Review Analysis ---
Average Rating (Sample Mean): 4.60
95% Confidence Interval: (4.58, 4.62)
Customer Satisfaction Level: High
```

EXP_23



EXP_23 To Analyze the data using hypothesis testing and calculate the p-value to determine if the new treatment has a statistically significant effect comp...
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind
```

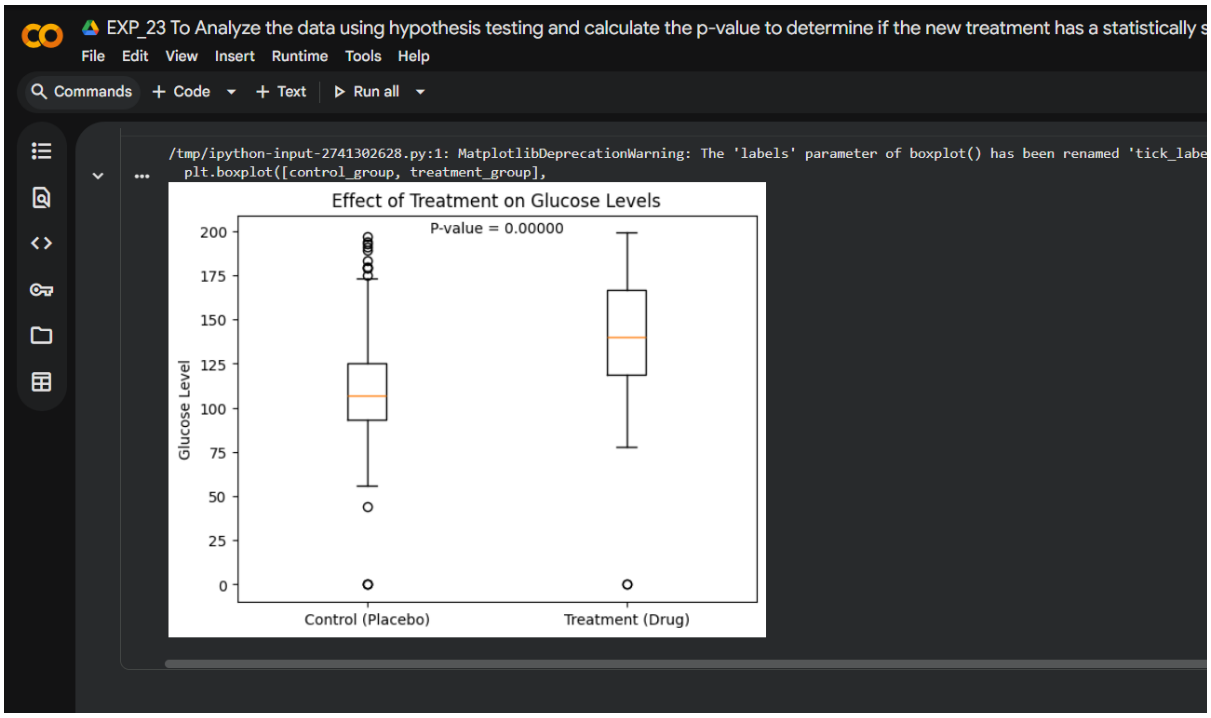
```python
df = pd.read_csv("diabetes.csv")
```

```python
control_group = df[df["Outcome"] == 0]["Glucose"]
treatment_group = df[df["Outcome"] == 1]["Glucose"]
```

```python
t_stat, p_value = ttest_ind(control_group, treatment_group)
print("T-statistic:", t_stat)
print("P-value:", p_value)
```

```
T-statistic: -14.600060005973894
P-value: 8.935431645289912e-43
```

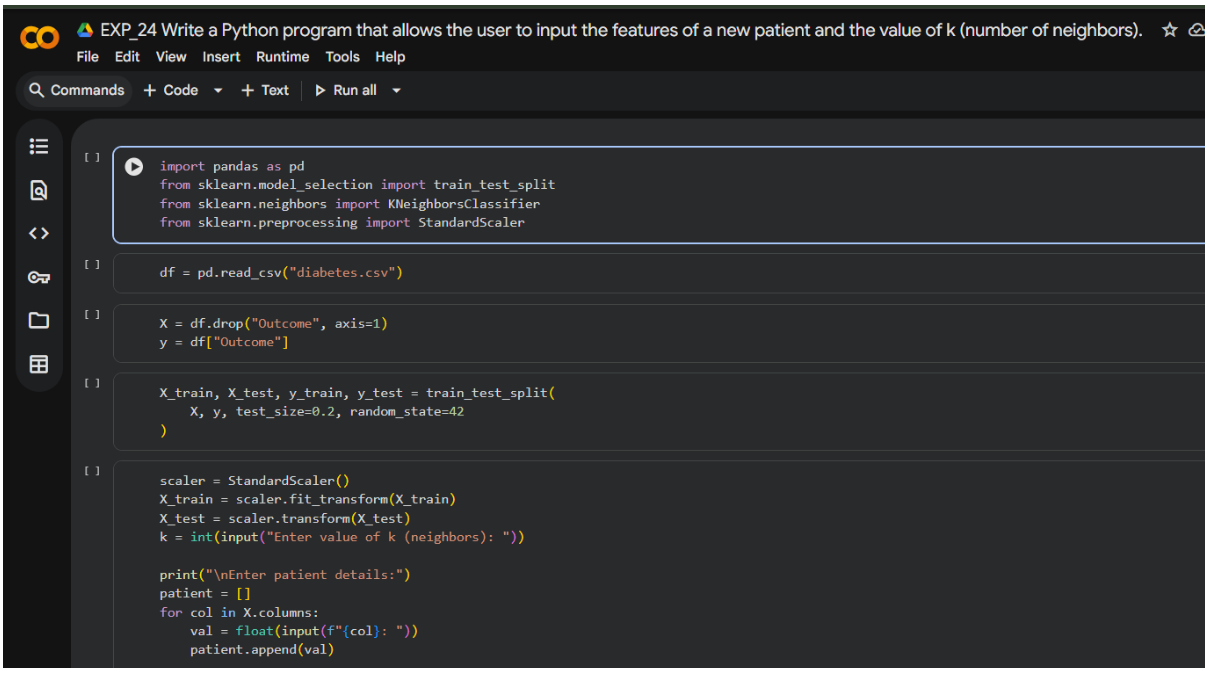
```python
plt.boxplot([control_group, treatment_group],
            labels=["Control (Placebo)", "Treatment (Drug)"])
plt.title("Effect of Treatment on Glucose Levels")
plt.ylabel("Glucose Level")
plt.text(1.5, max(df["Glucose"]), f"P-value = {p_value:.5f}", ha="center")
plt.show()
```


EXP_23 To Analyze the data using hypothesis testing and calculate the p-value to determine if the new treatment has a statistically s
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```
/tmp/ipython-input-2741302628.py:1: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labe
  plt.boxplot([control_group, treatment_group],
```

Effect of Treatment on Glucose Levels
P-value = 0.00000
Glucose Level
200
175
150
125
100
75
50
25
0
Control (Placebo)
Treatment (Drug)

EXP_24


EXP_24 Write a Python program that allows the user to input the features of a new patient and the value of k (number of neighbors).
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

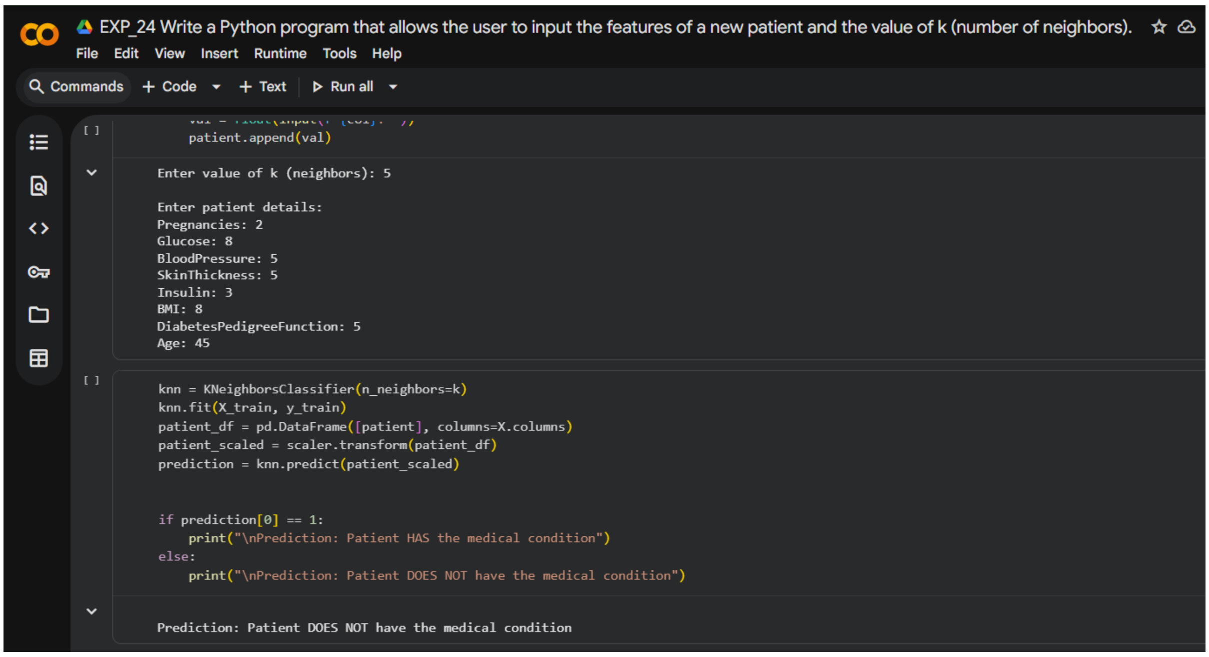
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
```

```
df = pd.read_csv("diabetes.csv")
```

```
X = df.drop("Outcome", axis=1)
y = df["Outcome"]
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
k = int(input("Enter value of k (neighbors): "))

print("\nEnter patient details:")
patient = []
for col in X.columns:
    val = float(input(f"{col}: "))
    patient.append(val)
```


EXP_24 Write a Python program that allows the user to input the features of a new patient and the value of k (number of neighbors).
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

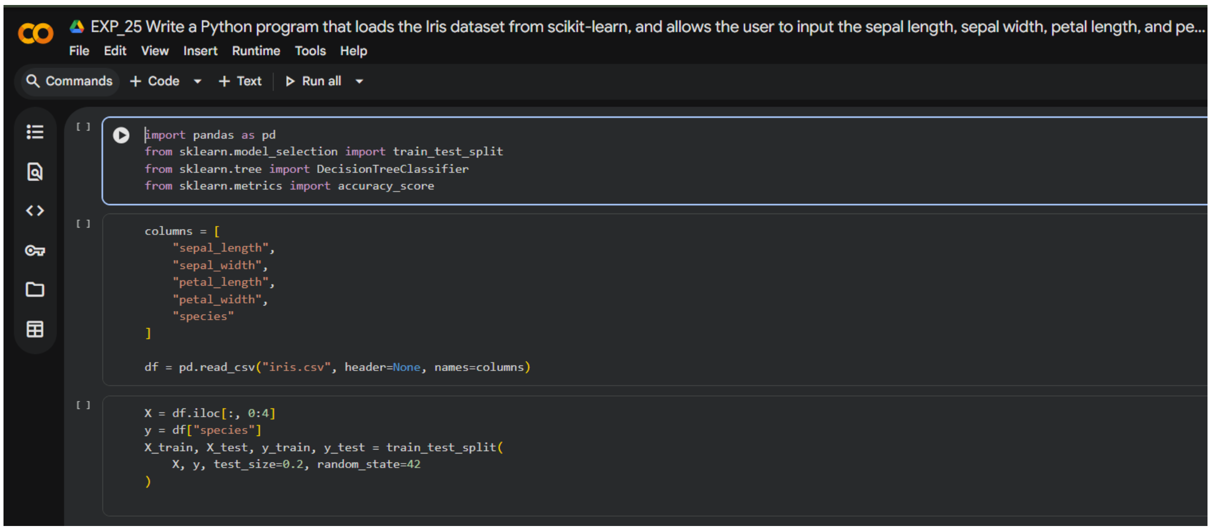
```
        patient.append(val)
```

```
Enter value of k (neighbors): 5

Enter patient details:
Pregnancies: 2
Glucose: 8
BloodPressure: 5
SkinThickness: 5
Insulin: 3
BMI: 8
DiabetesPedigreeFunction: 5
Age: 45
```

```
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
patient_df = pd.DataFrame([patient], columns=X.columns)
patient_scaled = scaler.transform(patient_df)
prediction = knn.predict(patient_scaled)


if prediction[0] == 1:
    print("\nPrediction: Patient HAS the medical condition")
else:
    print("\nPrediction: Patient DOES NOT have the medical condition")
```

```
Prediction: Patient DOES NOT have the medical condition
```

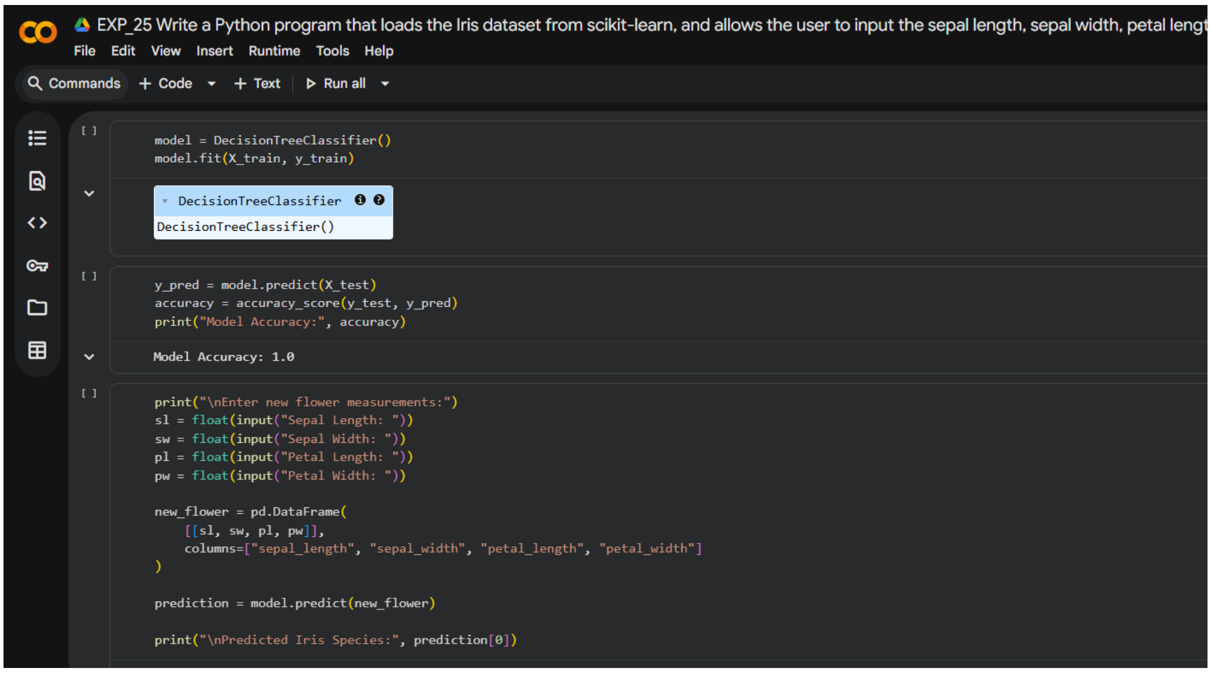
EXP_25


EXP_25 Write a Python program that loads the Iris dataset from scikit-learn, and allows the user to input the sepal length, sepal width, petal length, and pe...
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
columns = [
    "sepal_length",
    "sepal_width",
    "petal_length",
    "petal_width",
    "species"
]

df = pd.read_csv("iris.csv", header=None, names=columns)
```

```
X = df.iloc[:, 0:4]
y = df["species"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```


EXP_25 Write a Python program that loads the Iris dataset from scikit-learn, and allows the user to input the sepal length, sepal width, petal lengt
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

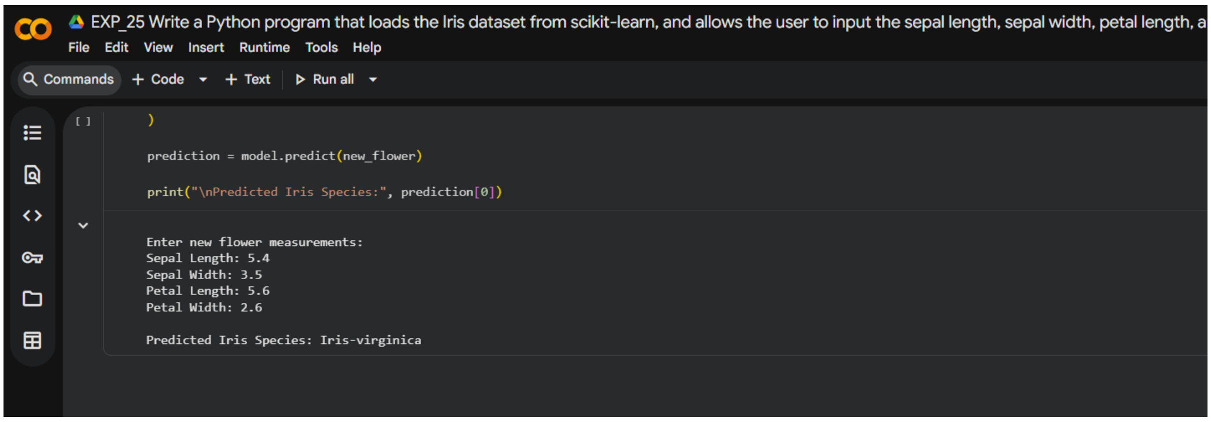
```
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
```

```
Model Accuracy: 1.0
```

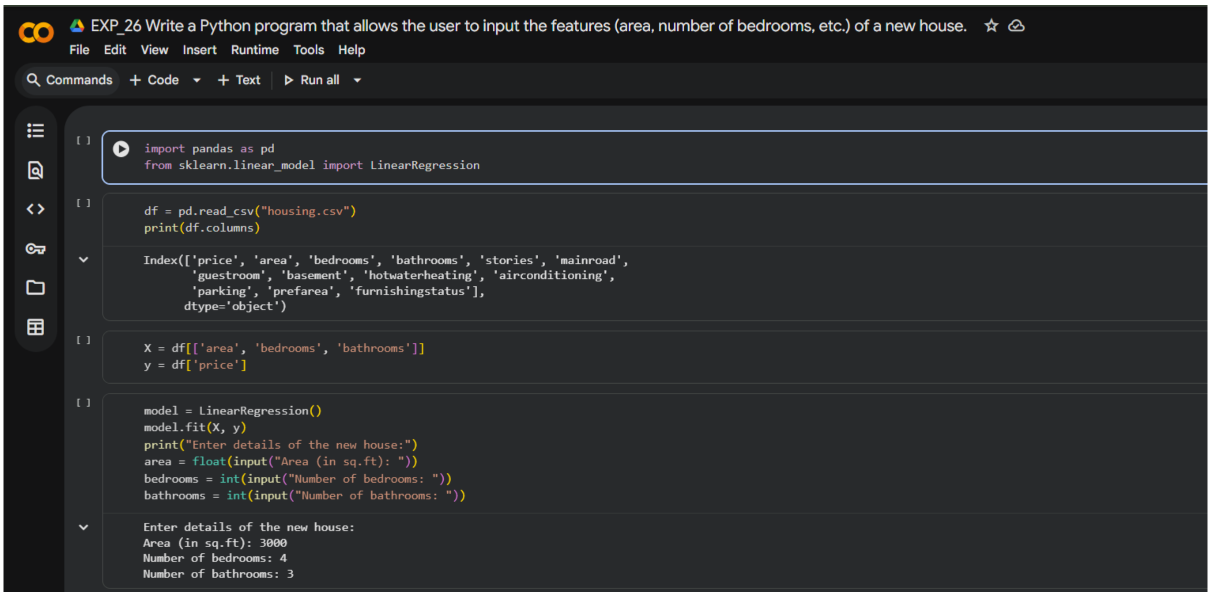
```
print("\nEnter new flower measurements:")
sl = float(input("Sepal Length: "))
sw = float(input("Sepal Width: "))
pl = float(input("Petal Length: "))
pw = float(input("Petal Width: "))

new_flower = pd.DataFrame(
    [[sl, sw, pl, pw]],
    columns=["sepal_length", "sepal_width", "petal_length", "petal_width"]
)

prediction = model.predict(new_flower)

print("\nPredicted Iris Species:", prediction[0])
```


EXP_25 Write a Python program that loads the Iris dataset from scikit-learn, and allows the user to input the sepal length, sepal width, petal length, a
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```
)

prediction = model.predict(new_flower)

print("\nPredicted Iris Species:", prediction[0])
```

```
Enter new flower measurements:
Sepal Length: 5.4
Sepal Width: 3.5
Petal Length: 5.6
Petal Width: 2.6

Predicted Iris Species: Iris-virginica
```

EXP_26


EXP_26 Write a Python program that allows the user to input the features (area, number of bedrooms, etc.) of a new house.
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```
import pandas as pd
from sklearn.linear_model import LinearRegression
```

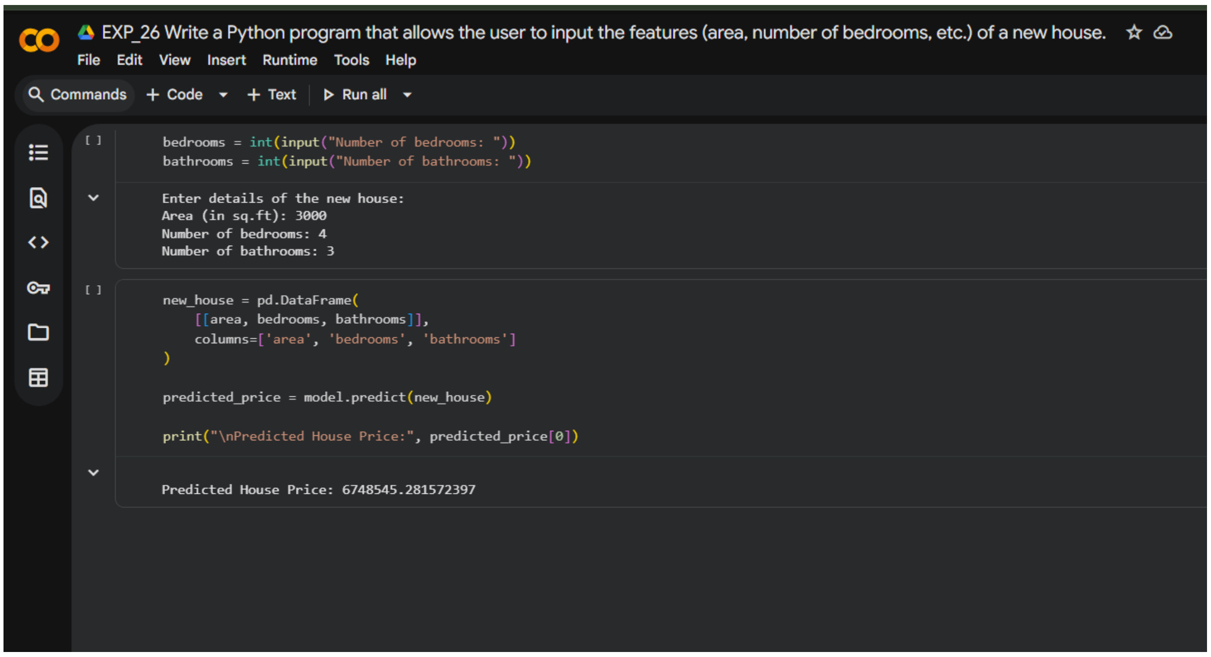
```
df = pd.read_csv("housing.csv")
print(df.columns)
```

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

```
X = df[['area', 'bedrooms', 'bathrooms']]
y = df['price']
```

```
model = LinearRegression()
model.fit(X, y)
print("Enter details of the new house:")
area = float(input("Area (in sq.ft): "))
bedrooms = int(input("Number of bedrooms: "))
bathrooms = int(input("Number of bathrooms: "))
```

```
Enter details of the new house:
Area (in sq.ft): 3000
Number of bedrooms: 4
Number of bathrooms: 3
```


EXP_26 Write a Python program that allows the user to input the features (area, number of bedrooms, etc.) of a new house.
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

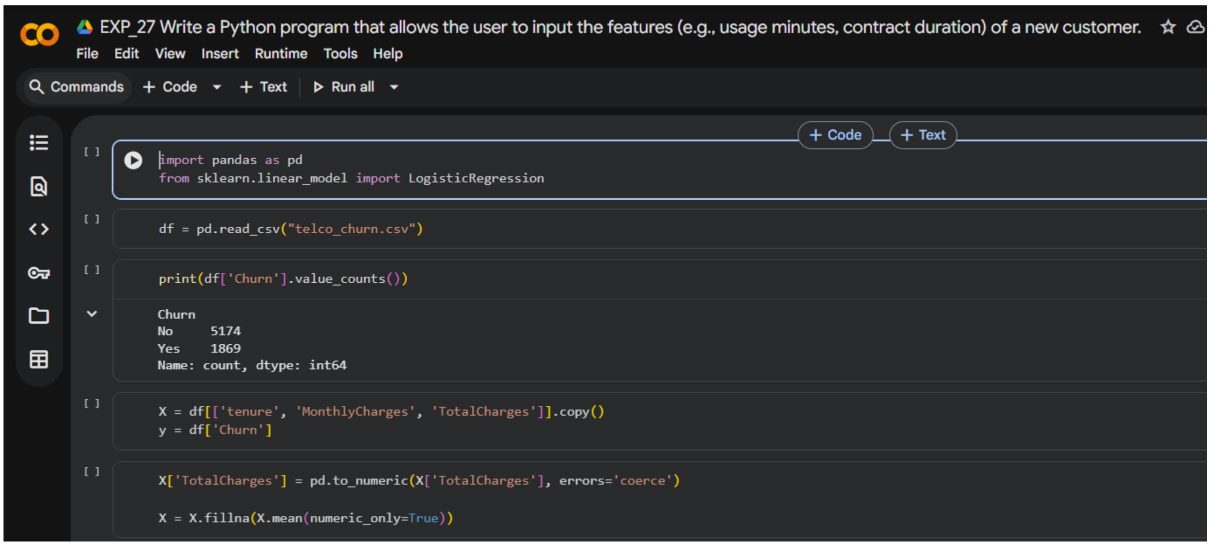
```python
bedrooms = int(input("Number of bedrooms: "))
bathrooms = int(input("Number of bathrooms: "))
```

```
Enter details of the new house:
Area (in sq.ft): 3000
Number of bedrooms: 4
Number of bathrooms: 3
```

```python
new_house = pd.DataFrame(
    [[area, bedrooms, bathrooms]],
    columns=['area', 'bedrooms', 'bathrooms']
)

predicted_price = model.predict(new_house)

print("\nPredicted House Price:", predicted_price[0])
```

```
Predicted House Price: 6748545.281572397
```

EXP_27


EXP_27 Write a Python program that allows the user to input the features (e.g., usage minutes, contract duration) of a new customer.
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
+ Code + Text

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
```

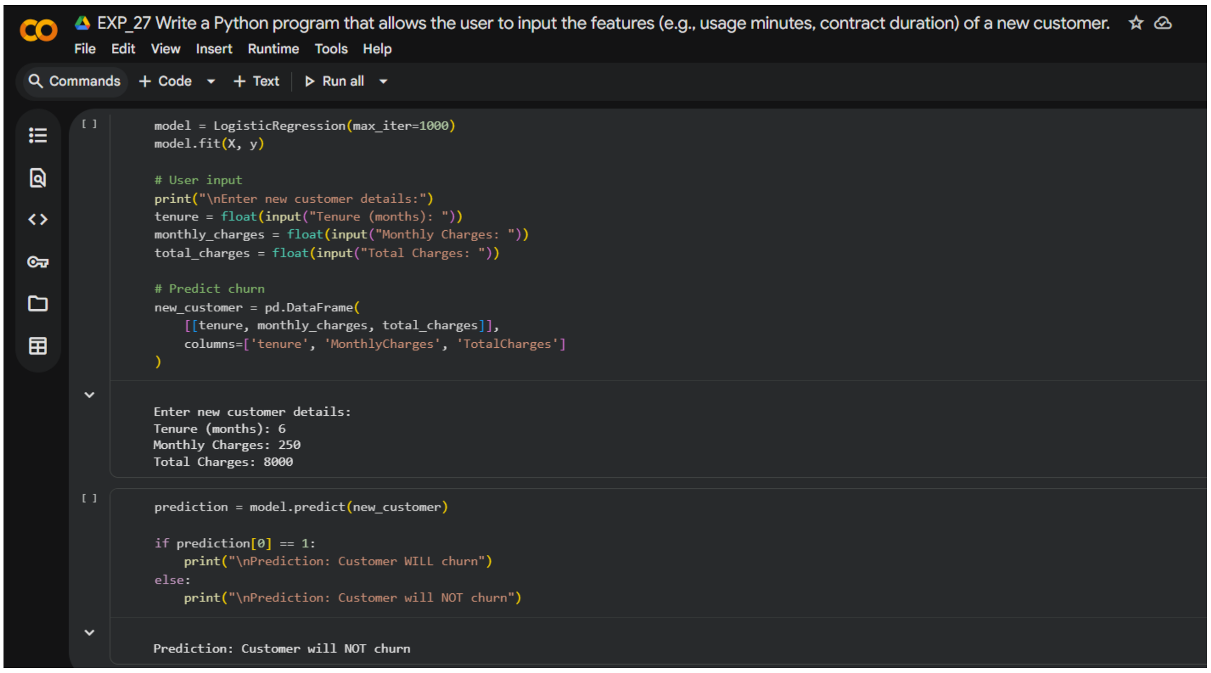
```python
df = pd.read_csv("telco_churn.csv")
```

```python
print(df['Churn'].value_counts())
```

```
Churn
No     5174
Yes    1869
Name: count, dtype: int64
```

```python
X = df[['tenure', 'MonthlyCharges', 'TotalCharges']].copy()
y = df['Churn']
```

```python
X['TotalCharges'] = pd.to_numeric(X['TotalCharges'], errors='coerce')

X = X.fillna(X.mean(numeric_only=True))
```


EXP_27 Write a Python program that allows the user to input the features (e.g., usage minutes, contract duration) of a new customer.
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

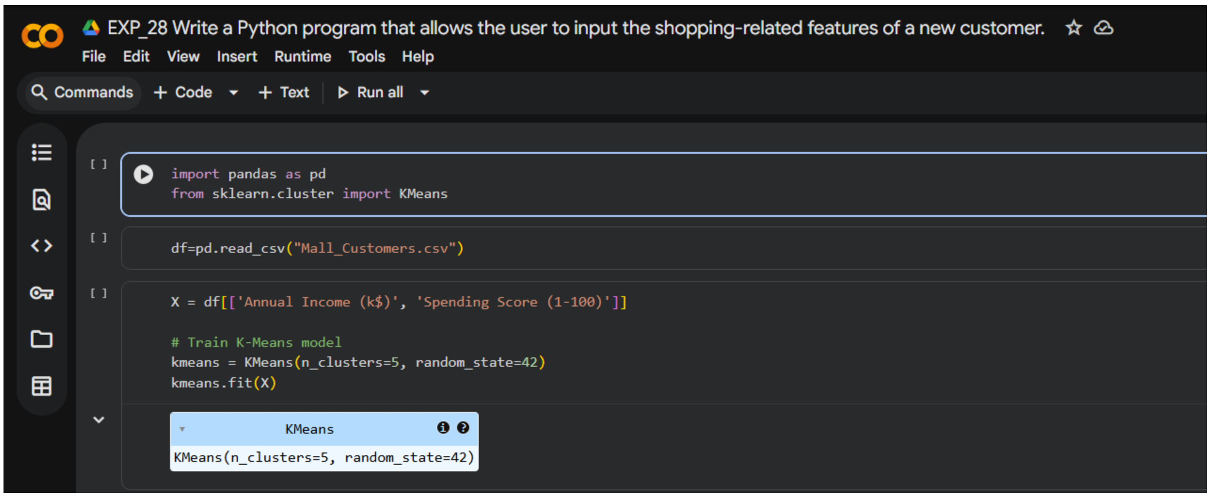
```
model = LogisticRegression(max_iter=1000)
model.fit(X, y)

# User input
print("\nEnter new customer details:")
tenure = float(input("Tenure (months): "))
monthly_charges = float(input("Monthly Charges: "))
total_charges = float(input("Total Charges: "))

# Predict churn
new_customer = pd.DataFrame(
    [[tenure, monthly_charges, total_charges]],
    columns=['tenure', 'MonthlyCharges', 'TotalCharges']
)
```

```
Enter new customer details:
Tenure (months): 6
Monthly Charges: 250
Total Charges: 8000
```

```
prediction = model.predict(new_customer)

if prediction[0] == 1:
    print("\nPrediction: Customer WILL churn")
else:
    print("\nPrediction: Customer will NOT churn")
```

```
Prediction: Customer will NOT churn
```

EXP_28

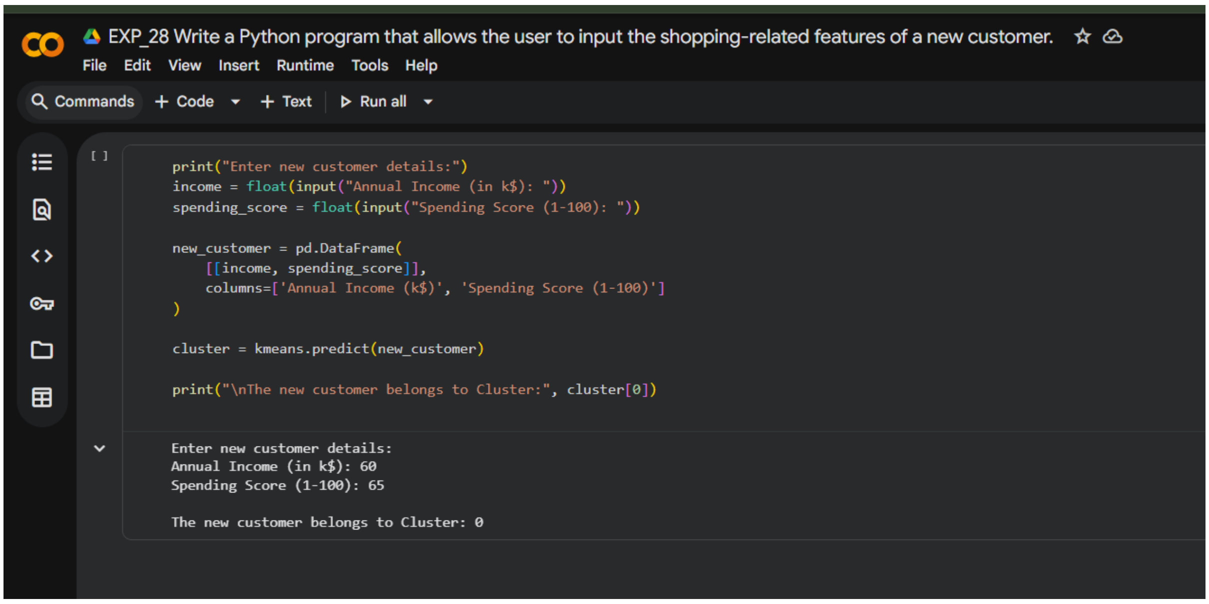

EXP_28 Write a Python program that allows the user to input the shopping-related features of a new customer.
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```
import pandas as pd
from sklearn.cluster import KMeans
```

```
df=pd.read_csv("Mall_Customers.csv")
```

```
X = df[['Annual Income (k$)', 'Spending Score (1-100)']]

# Train K-Means model
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans.fit(X)
```

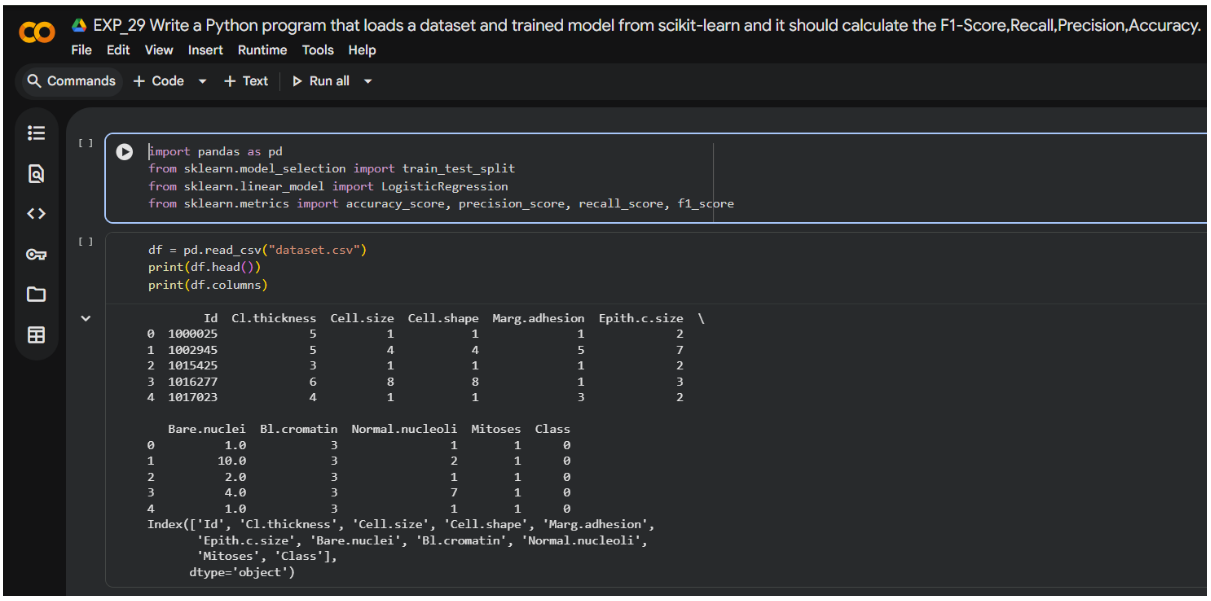
```
KMeans
KMeans(n_clusters=5, random_state=42)
```


EXP_28 Write a Python program that allows the user to input the shopping-related features of a new customer.
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all

```
print("Enter new customer details:")
income = float(input("Annual Income (in k$): "))
spending_score = float(input("Spending Score (1-100): "))

new_customer = pd.DataFrame(
    [[income, spending_score]],
    columns=['Annual Income (k$)', 'Spending Score (1-100)']
)

cluster = kmeans.predict(new_customer)

print("\nThe new customer belongs to Cluster:", cluster[0])
```

```
Enter new customer details:
Annual Income (in k$): 60
Spending Score (1-100): 65

The new customer belongs to Cluster: 0
```

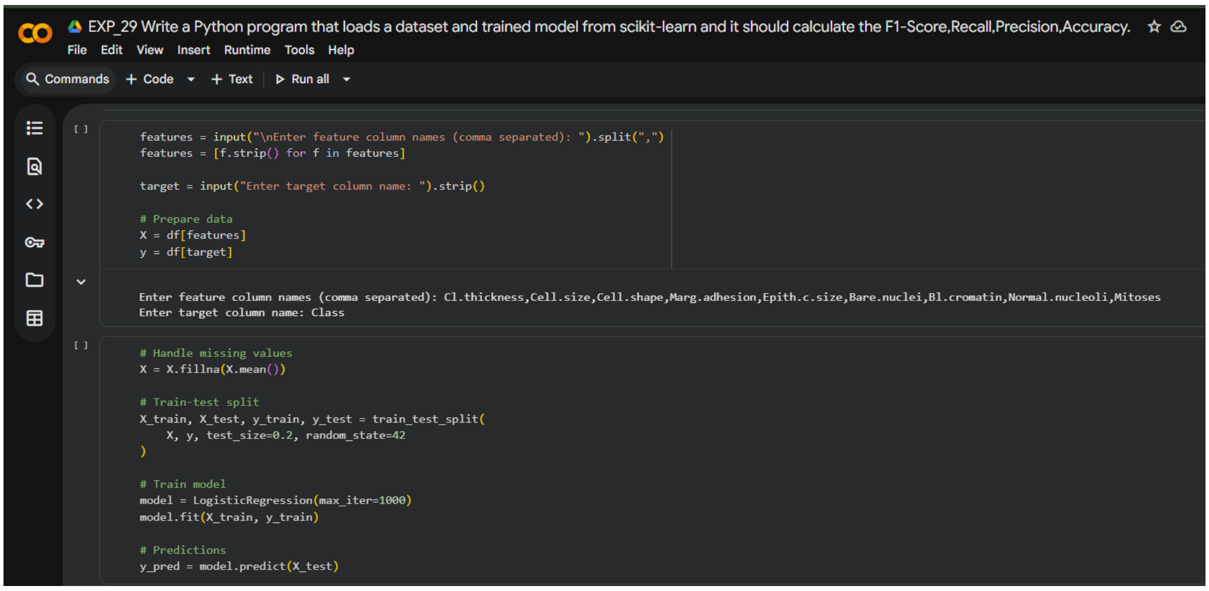
EXP_29


EXP_29 Write a Python program that loads a dataset and trained model from scikit-learn and it should calculate the F1-Score,Recall,Precision,Accuracy.
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
df = pd.read_csv("dataset.csv")
print(df.head())
print(df.columns)
```

```
        Id  Cl.thickness  Cell.size  Cell.shape  Marg.adhesion  Epith.c.size  \
0  1000025             5          1           1              1             2
1  1002945             5          4           4              5             7
2  1015425             3          1           1              1             2
3  1016277             6          8           8              1             3
4  1017023             4          1           1              3             2

   Bare.nuclei  Bl.cromatin  Normal.nucleoli  Mitoses  Class
0          1.0            3                1        1      0
1         10.0            3                2        1      0
2          2.0            3                1        1      0
3          4.0            3                7        1      0
4          1.0            3                1        1      0
Index(['Id', 'Cl.thickness', 'Cell.size', 'Cell.shape', 'Marg.adhesion',
       'Epith.c.size', 'Bare.nuclei', 'Bl.cromatin', 'Normal.nucleoli',
       'Mitoses', 'Class'],
      dtype='object')
```

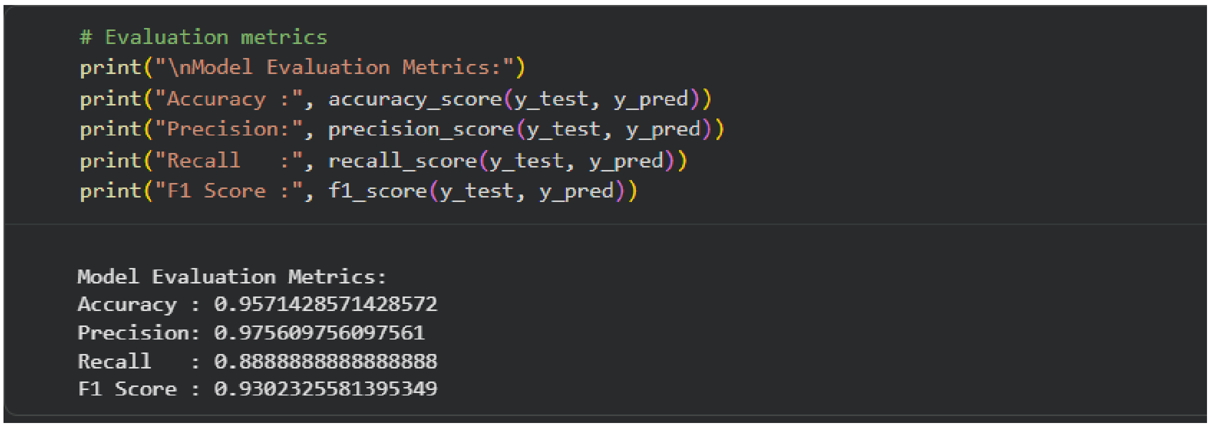

EXP_29 Write a Python program that loads a dataset and trained model from scikit-learn and it should calculate the F1-Score,Recall,Precision,Accuracy.

File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```python
features = input("\nEnter feature column names (comma separated): ").split(",")
features = [f.strip() for f in features]

target = input("Enter target column name: ").strip()

# Prepare data
X = df[features]
y = df[target]
```

```
Enter feature column names (comma separated): Cl.thickness,Cell.size,Cell.shape,Marg.adhesion,Epith.c.size,Bare.nuclei,Bl.cromatin,Normal.nucleoli,Mitoses
Enter target column name: Class
```

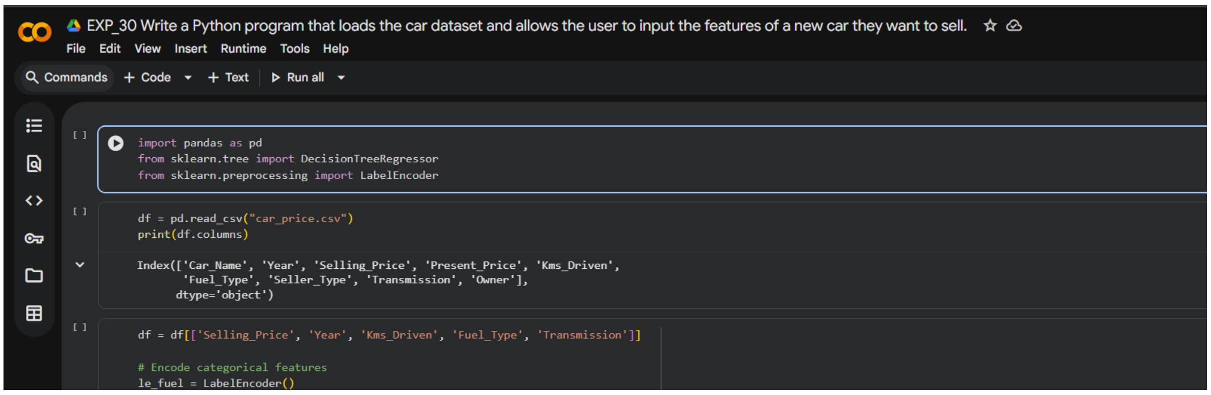
```python
# Handle missing values
X = X.fillna(X.mean())

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

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



```python
# Evaluation metrics
print("\nModel Evaluation Metrics:")
print("Accuracy :", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall    :", recall_score(y_test, y_pred))
print("F1 Score :", f1_score(y_test, y_pred))
```

```
Model Evaluation Metrics:
Accuracy : 0.9571428571428572
Precision: 0.975609756097561
Recall    : 0.8888888888888888
F1 Score : 0.9302325581395349
```

EXP_30


EXP_30 Write a Python program that loads the car dataset and allows the user to input the features of a new car they want to sell.

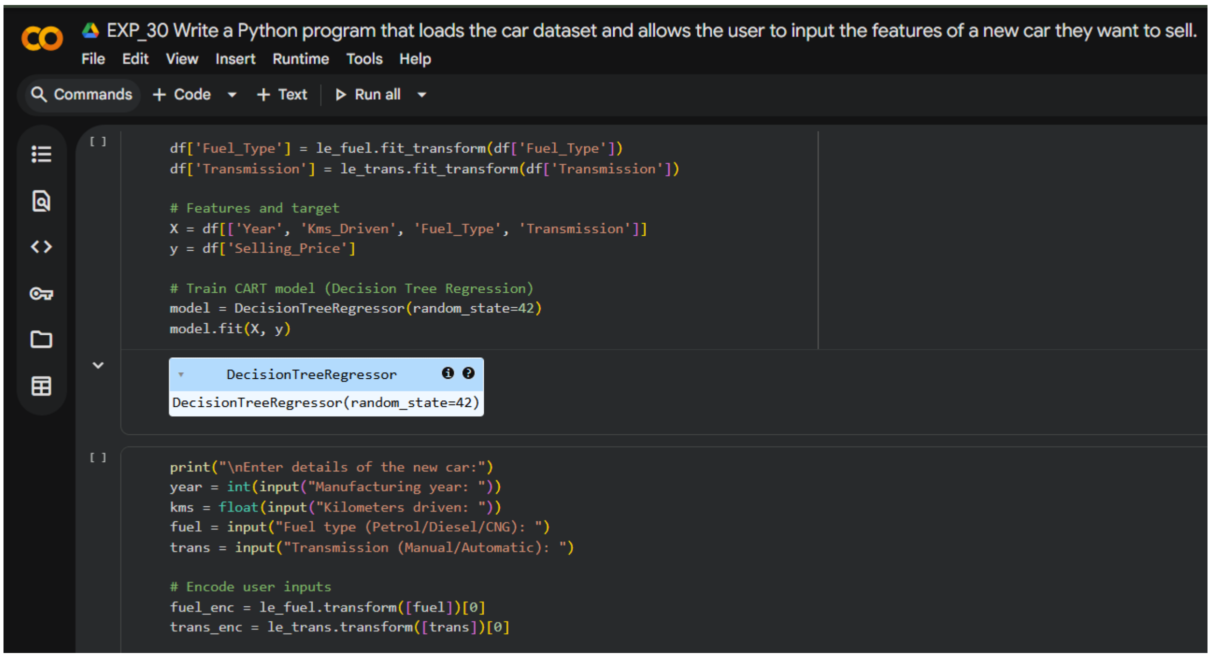
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```python
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import LabelEncoder
```

```python
df = pd.read_csv("car_price.csv")
print(df.columns)
```

```
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner'],
      dtype='object')
```

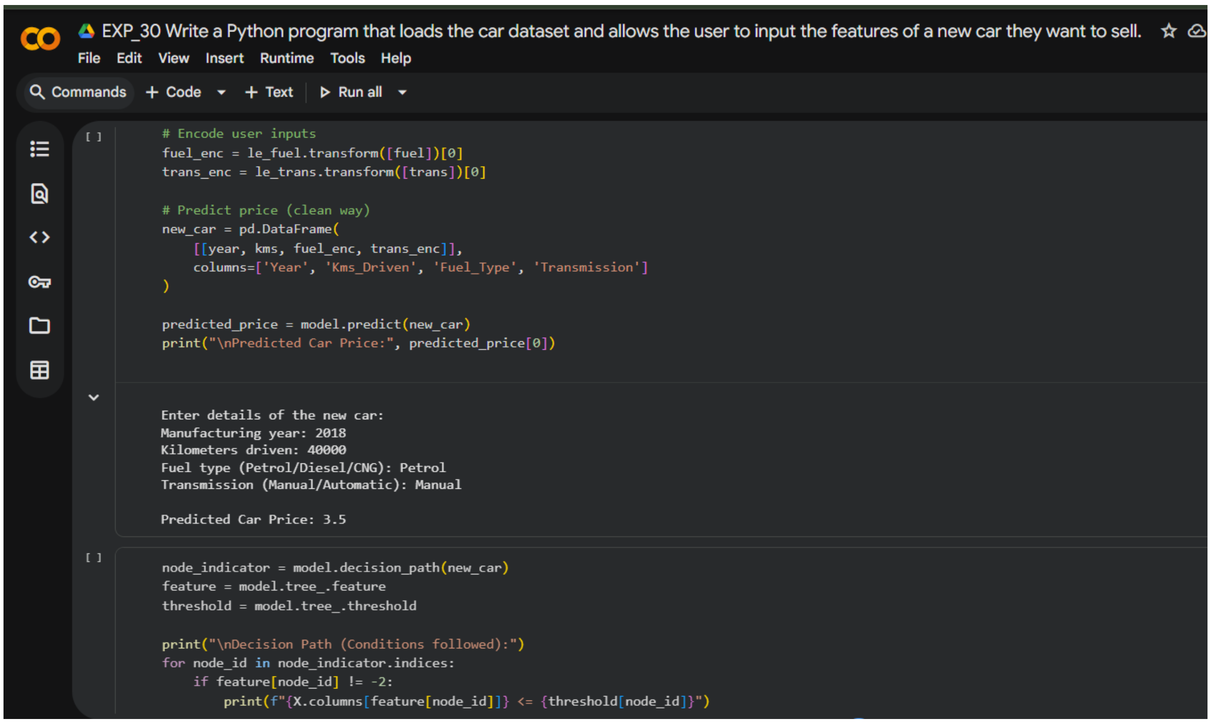
```python
df = df[['Selling_Price', 'Year', 'Kms_Driven', 'Fuel_Type', 'Transmission']]

# Encode categorical features
le_fuel = LabelEncoder()
```


EXP_30 Write a Python program that loads the car dataset and allows the user to input the features of a new car they want to sell.
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```python
df['Fuel_Type'] = le_fuel.fit_transform(df['Fuel_Type'])
df['Transmission'] = le_trans.fit_transform(df['Transmission'])

# Features and target
X = df[['Year', 'Kms_Driven', 'Fuel_Type', 'Transmission']]
y = df['Selling_Price']

# Train CART model (Decision Tree Regression)
model = DecisionTreeRegressor(random_state=42)
model.fit(X, y)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

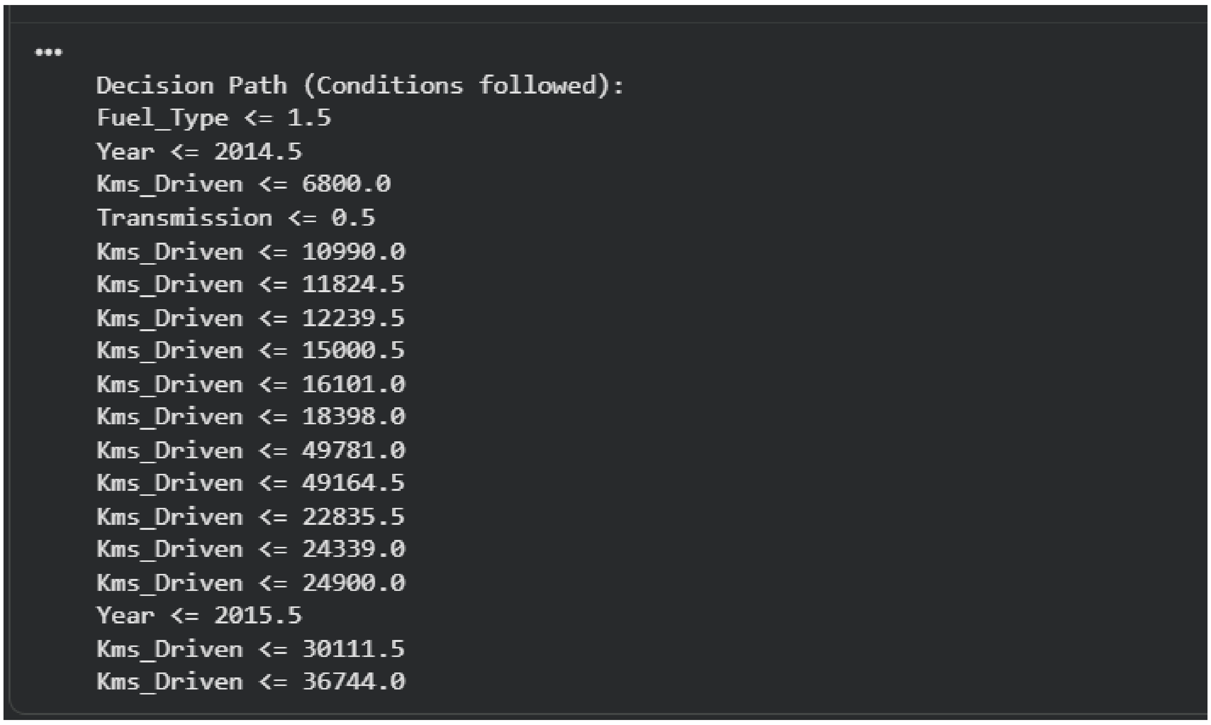
```python
print("\nEnter details of the new car:")
year = int(input("Manufacturing year: "))
kms = float(input("Kilometers driven: "))
fuel = input("Fuel type (Petrol/Diesel/CNG): ")
trans = input("Transmission (Manual/Automatic): ")

# Encode user inputs
fuel_enc = le_fuel.transform([fuel])[0]
trans_enc = le_trans.transform([trans])[0]
```


EXP_30 Write a Python program that loads the car dataset and allows the user to input the features of a new car they want to sell.
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

```python
# Encode user inputs
fuel_enc = le_fuel.transform([fuel])[0]
trans_enc = le_trans.transform([trans])[0]

# Predict price (clean way)
new_car = pd.DataFrame(
    [[year, kms, fuel_enc, trans_enc]],
    columns=['Year', 'Kms_Driven', 'Fuel_Type', 'Transmission']
)

predicted_price = model.predict(new_car)
print("\nPredicted Car Price:", predicted_price[0])
```

```
Enter details of the new car:
Manufacturing year: 2018
Kilometers driven: 40000
Fuel type (Petrol/Diesel/CNG): Petrol
Transmission (Manual/Automatic): Manual

Predicted Car Price: 3.5
```

```python
node_indicator = model.decision_path(new_car)
feature = model.tree_.feature
threshold = model.tree_.threshold

print("\nDecision Path (Conditions followed):")
for node_id in node_indicator.indices:
    if feature[node_id] != -2:
        print(f"{X.columns[feature[node_id]]} <= {threshold[node_id]}")
```



```
...
    Decision Path (Conditions followed):
    Fuel_Type <= 1.5
    Year <= 2014.5
    Kms_Driven <= 6800.0
    Transmission <= 0.5
    Kms_Driven <= 10990.0
    Kms_Driven <= 11824.5
    Kms_Driven <= 12239.5
    Kms_Driven <= 15000.5
    Kms_Driven <= 16101.0
    Kms_Driven <= 18398.0
    Kms_Driven <= 49781.0
    Kms_Driven <= 49164.5
    Kms_Driven <= 22835.5
    Kms_Driven <= 24339.0
    Kms_Driven <= 24900.0
    Year <= 2015.5
    Kms_Driven <= 30111.5
    Kms_Driven <= 36744.0
```