Analyze Traffic Safety Data with Python

Try some of these resources for extra help as you work:

- View the Analyze Traffic Safety Data with Python cheatsheet
- View the solution notebook
- Learn more about analyzing traffic safety data in this introductory article

In [1]:
```
import pandas as pd
import datetime as dt
from scipy.stats import pearsonr
from sklearn.linear_model import LinearRegression
import seaborn as sns
import matplotlib.pyplot as plt


%matplotlib inline
# set plot theme and palette
sns.set_theme()
sns.set_palette('colorblind')
```

Traffic data exploration

1. Inspect the traffic safety dataset

After running the first cell to load all necessary libraries, we need to load our dataset. Using pandas, load the dataset `traffic.csv` and save it as `traffic`. Inspect the first few rows.

In [2]:
```
# load dataset
## YOUR CODE HERE ##
traffic = pd.read_csv('traffic.csv')

# inspect first few rows
## YOUR CODE HERE ##
traffic.head()
```

Out[2]:

	Date	Crashes_per_100k	Season
0	2006-01-01	169.176541	Winter
1	2006-02-01	154.028836	Winter
2	2006-03-01	159.930002	Spring
3	2006-04-01	155.741270	Spring
4	2006-05-01	168.179208	Spring

2. Inspect and format data types

The `traffic` data frame contains three columns: `Date`, `Crashes_per_100k`, and `Season`. In order to plot the `Crashes_per_100k` column as a time series, we need to make sure that the `Date` column is in date format. Inspect the data types in the data frame, convert the `Date` column to date format, and inspect the data types a second time.

In [3]:
```
# inspect data types
## YOUR CODE HERE ##
traffic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180 entries, 0 to 179
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Date              180 non-null    object
 1   Crashes_per_100k  180 non-null    float64
 2   Season            180 non-null    object
dtypes: float64(1), object(2)
memory usage: 4.3+ KB
```

▶ *What did we discover in this step? Toggle to check!*

Convert the `Date` column to the date datatype using the `pd.to_datatime(column)` function.

In [4]:
```
# convert Date to date format
## YOUR CODE HERE ##
traffic['Date'] = pd.to_datetime(traffic['Date'])

# inspect data types
## YOUR CODE HERE ##
traffic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180 entries, 0 to 179
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Date              180 non-null    datetime64[ns]
 1   Crashes_per_100k  180 non-null    float64
 2   Season            180 non-null    object
dtypes: datetime64[ns](1), float64(1), object(1)
memory usage: 4.3+ KB
```

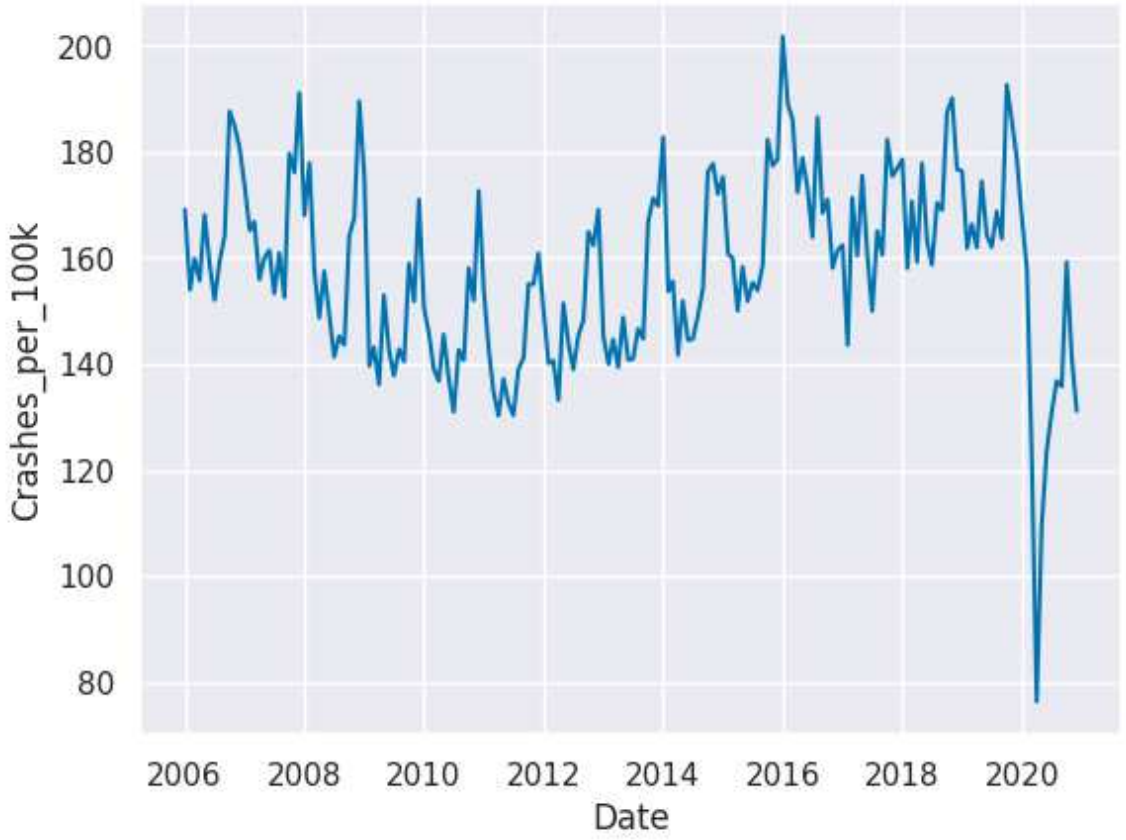
▶ *What did we discover in this step? Toggle to check!*

3. Visualize traffic safety data

To get a sense of trends that may exist in the data, use seaborn's `sns.lineplot()` function to create a line plot of the `traffic` data with `Date` on the x-axis and `Crashes_per_100k` on the y-axis.

In [5]:
```
# create line plot
## YOUR CODE HERE ##
```

```
sns.lineplot(x = 'Date', y='Crashes_per_100k', data = traffic)
plt.show()
```



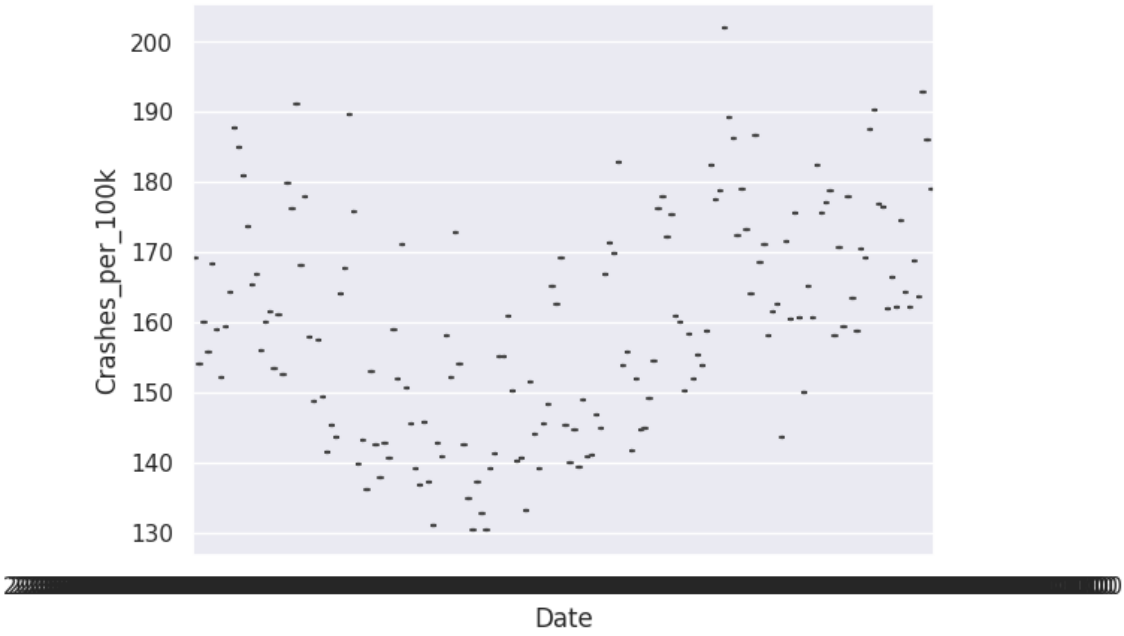
▶ *What did we discover in this step? Toggle to check!*

4. Visualize seasonal rates

Since we saw a fair amount of variance in the number of collisions occurring throughout the year, we might hypothesize that the number of collisions increases or decreases during different seasons. We can visually explore this with a box plot.

Use `sns.boxplot()` with crash rate on the x-axis and season on the y-axis. Remove the anomolous 2020 data by adjusting the `data` parameter to `traffic[traffic.Date.dt.year != 2020]`.

In [6]:
```
# create box plot by season
## YOUR CODE HERE ##
sns.boxplot(x = 'Date', y='Crashes_per_100k', data = traffic[traffic.Date.dt.yea
plt.show()
```



▶ *What did we discover in this step? Toggle to check!*

Smartphone data exploration

5. Inspect the smartphone use dataset

The dataset `crashes_smartphones.csv` contains smartphone data from Pew Research Center matched to normalized crash rates from the `traffic` data frame for the years 2011 to 2019.

▶ **Toggle for an overview of the variables in this dataset.**

Load the dataset as `smartphones` and inspect the first few rows.

In [7]:
```
# import dataset
## YOUR CODE HERE ##
smartphones = pd.read_csv('crashes_smartphones.csv')

# inspect first few rows
## YOUR CODE HERE ##
smartphones.head()
```

Out[7]:

	Month_Year	Crashes_per_100k	Season	Smartphone_Survey_Date	Smartphone_usage
0	Apr-12	133.213685	Spring	4/3/12	46
1	Apr-15	150.077792	Spring	4/12/15	67
2	Apr-16	172.401948	Spring	4/4/16	72
3	Aug-12	145.403147	Summer	8/5/12	44
4	Dec-12	169.160811	Winter	12/9/12	45

6. Format date data type

Similar to the `traffic` data frame, the `smartphones` data frame has a date column that is not properly formatted. Convert the `Smartphone_Survey_Date` column to the date data type using the `pd.to_datetime()` function and then inspect the data types in the data frame.

In [8]:
```
# convert Date to date format
## YOUR CODE HERE ##
smartphones['Smartphone_Survey_Date'] = pd.to_datetime(smartphones['Smartphone_S

# inspect data types
## YOUR CODE HERE ##
smartphones.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28 entries, 0 to 27
Data columns (total 5 columns):
 #   Column                 Non-Null Count  Dtype         
---  ------                 --------------  -----         
 0   Month_Year             28 non-null     object        
 1   Crashes_per_100k       28 non-null     float64       
 2   Season                 28 non-null     object        
 3   Smartphone_Survey_Date  28 non-null     datetime64[ns]
 4   Smartphone_usage       28 non-null     int64         
dtypes: datetime64[ns](1), float64(1), int64(1), object(2)
memory usage: 1.2+ KB
```

change to datetime object

YOUR CODE HERE

smartphones['Smartphone_Survey_Date'] = pd.to_datetime(smartphones['Smartphone_Survey_Date'])

inspect data types

YOUR CODE HERE

smartphones.info()

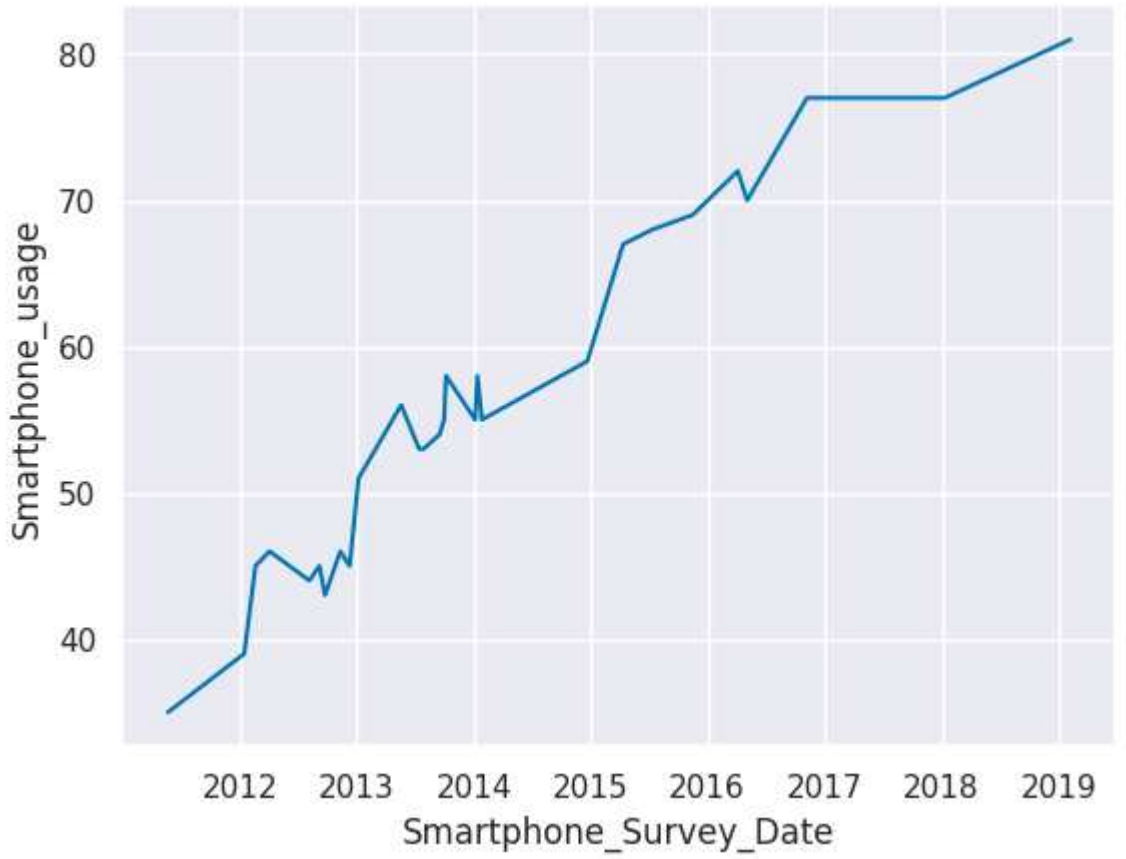
▶ *What did we discover in this step? Toggle to check!*

7. Visualize smartphone use data

Now let's take a look at smartphone use over time. Create a line plot of the `smartphones` data with `Smartphone_Survey_Date` on the x-axis and `Smartphone_usage` on the y-axis.

In [9]:
```
# create line plot
## YOUR CODE HERE ##
```

```
sns.lineplot(x = 'Smartphone_Survey_Date', y='Smartphone_usage', data = smartphc
plt.show()
```



▶ *What did we discover in this step? Toggle to check!*

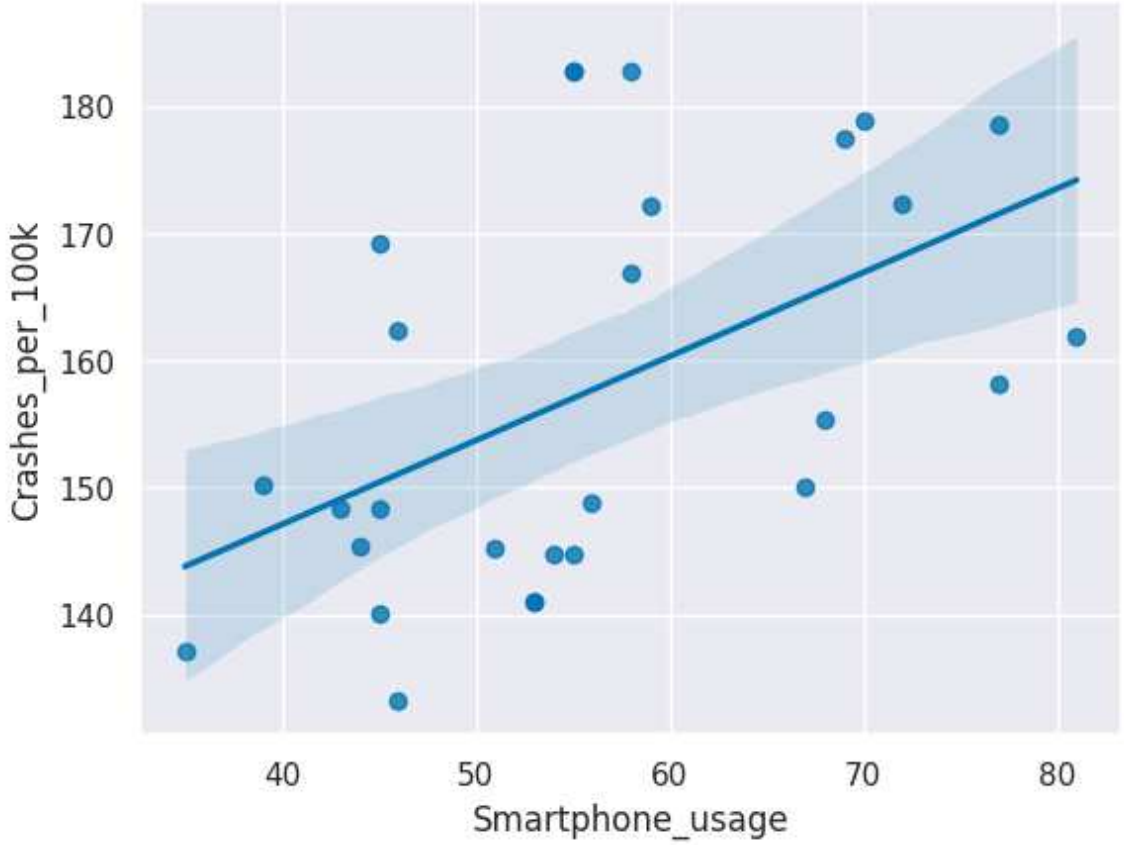
Relationship exploration

8. Visualize crash rate by smartphone use

A scatter plot with smartphone usage on one axis and crash rates on the other axis will give us an idea of whether there is a relationship between these two variables.

Create a scatter plot with a regression line using seaborn's `sns.regplot()` with `Smartphone_usage` on the x-axis and `Crashes_per_100k` on the y-axis.

In [10]:
```
# create scatter plot with regression line
## YOUR CODE HERE ##
sns.regplot(x = 'Smartphone_usage', y = 'Crashes_per_100k', data = smartphones)
plt.show()
```



▶ *What did we discover in this step? Toggle to check!*

9. Check the correlation coefficient

To test whether the correlation between `Smartphone_usage` and `Crashes_per_100k` is statistically significant, we can calculate the Pearson's *r* correlation coefficient and the associated *p*-value.

Use `corr, p = pearsonr(column1, column2)` on the `Smartphone_usage` and `Crashes_per_100k` columns in the `smartphones` dataframe. Then use the provided code to print `corr` and `p` to see the results.

In [11]:
```
# find Pearson's r and p-value
corr, p = pearsonr(smartphones.Smartphone_usage, smartphones.Crashes_per_100k)
## YOUR CODE HERE ## pearsonr(df.column1, df.column2)

# print corr and p
print("Pearson's r =",  round(corr,3))
print("p = ", round(p,3))
```

```
Pearson's r = 0.513
p =  0.005
```

▶ *What did we discover in this step? Toggle to check!*

Analysis

10. Run a linear regression

We can use a linear regression to predict crash rates based on smart phone usage. Let's regress crash rates on smartphone usage. Then we can predict the crash rate in 2020 and see if it matches the actual crash rate in 2020!

We have provided the code to convert the variables to NumPy arrays that will work with the modeling function. The `Smartphone_usage` array is saved as `X`, and the `Crashes_per_100k` array is saved as `y`.

Initiate the model by saving `LinearRegression()` to the variable `lm`. Then fit the model and run the regression with `.fit()`.

In [20]:
```
# convert columns to arrays
X = smartphones['Smartphone_usage'].to_numpy().reshape(-1, 1)
y = smartphones['Crashes_per_100k'].to_numpy().reshape(-1, 1)
lm = LinearRegression()
lm.fit(X,y)
```

Out[20]:
```
▾ LinearRegression
LinearRegression()
```

initiate the linear regression model

YOUR CODE HERE

lm = LinearRegression()

fit the model

YOUR CODE HERE

lm.fit(X, y)

11. Print and interpret regression coefficients

Let's see the values our model produced. Print the coefficients from our `lm` model. Then think about which parts of the regression line equation these values represent.

In [21]:
```
# print the coefficients
## YOUR CODE HERE ##
print("Coefficients: \n",lm.intercept_, lm.coef_)
```

```
Coefficients:
 [120.6637106] [[0.66103316]]
```

▶ *What did we discover in this step? Toggle to check!*

12. Make a prediction

Let's assume smartphone usage was the same for 2020 as it was for 2019. This is a reasonable asssumption since the increase in smartphone usage that we observed in our plot started to plateau at the end of the time series. Let's use this approximation and our regression model to predict the crash rate in 2020.

From our model output, the regression line equation is `Crashes_per_100k = 120.6637 + (0.6610 * Smartphone_usage)`. Run the provided code to view the smartphone usage rate for 2019. Then substitute this value into the equation, using Python as a calculator to predict the crash rate for 2020.

In [24]:
```
# get the smartphone usage rate from 2019
smartphones[smartphones['Month_Year'] == "Feb-19"].Smartphone_usage
```

Out[24]:
```
7    81
Name: Smartphone_usage, dtype: int64
```

In [29]:
```
# predict the crash rate in 2020 using the regression equation
## YOUR CODE HERE ##
Crashes_per_100k = 120.6637 + (0.6610 * smartphones.Smartphone_usage)
```

▶ *What did we discover in this step? Toggle to check!*

13. Compare to the actual rate

How good was our prediction? Get the actual crash rate for February of 2020 from the `traffic` dataframe using `pd.to_datetime("2020-02-01")` as the value for `Date`.

In [32]:
```
# get the actual crash rate in Feb 2020
## YOUR CODE HERE ##
traffic[traffic['Date'] == pd.to_datetime("2020-02-01")].Crashes_per_100k
```

Out[32]:
```
169    157.88955
Name: Crashes_per_100k, dtype: float64
```

▶ *What did we discover in this step? Toggle to check!*

14. Visualize the prediction

Let's plot our regression plot again, but let's add two new points on top:

- The predicted 2020 crash rate
- The actual 2020 crash rate

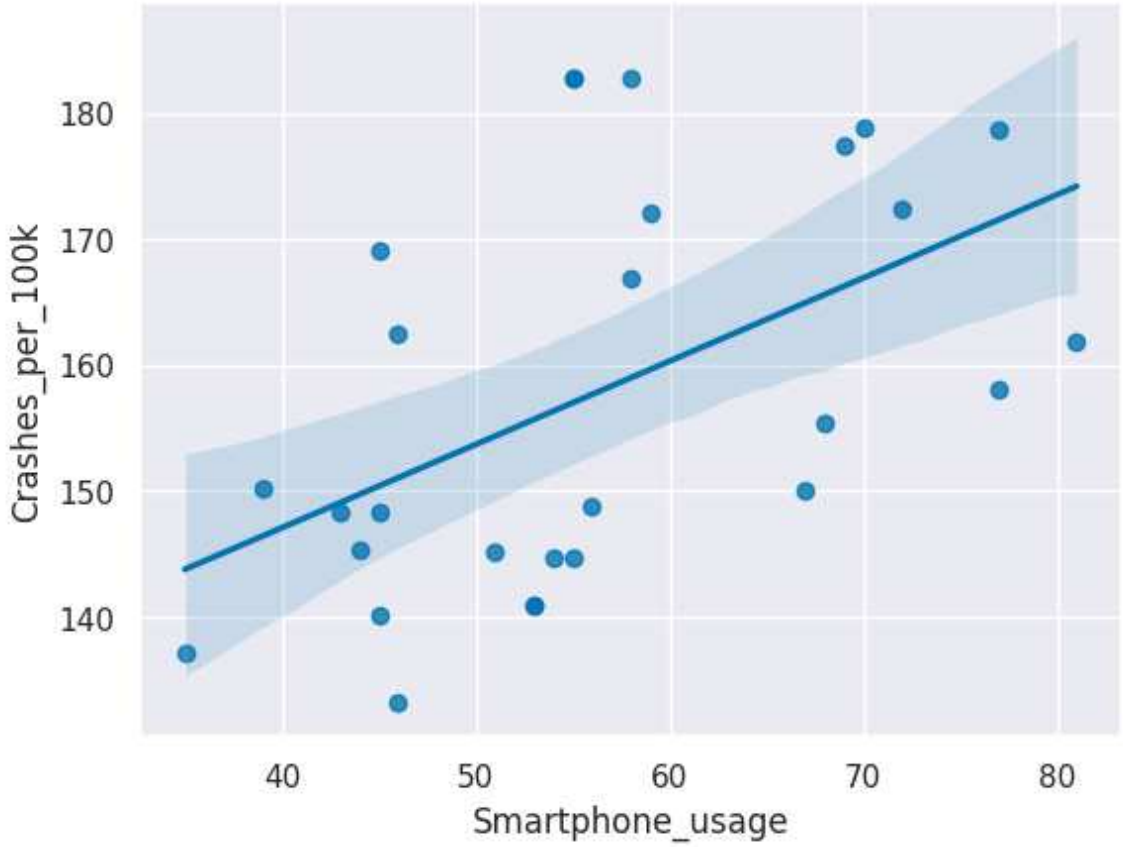
Code has been provided for the original regression plot and a legend title.

Add a scatter plot layer to add the 2020 predicted and actual crash rates that both used the 2019 smartphone usage rate. Use different colors and marker shapes for the predicted and actual 2020 crash rates.

In [47]:
```
# recreate the regression plot we made earlier
sns.regplot(x = 'Smartphone_usage', y = 'Crashes_per_100k', data = smartphones)


# add a scatter plot layer to show the actual and predicted 2020 values
## YOUR CODE HERE ##
#sns.scatterplot(x = [x1,x2], y = [y1,y2],
#hue = ['predicted','actual'] , style = ['predicted','actual'],
#markers = ['X','o'], palette=['navy','orange'], s=200)

# add legend title
#plt.legend(title='2020')
plt.show()
```



▶ *What did we discover in this step? Toggle to check!*