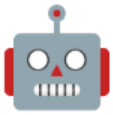


Microsoft machine learning

Created	@December 9, 2023 6:52 AM
Lesson Date	@December 9, 2023
Status	In Progress
Type	Lesson

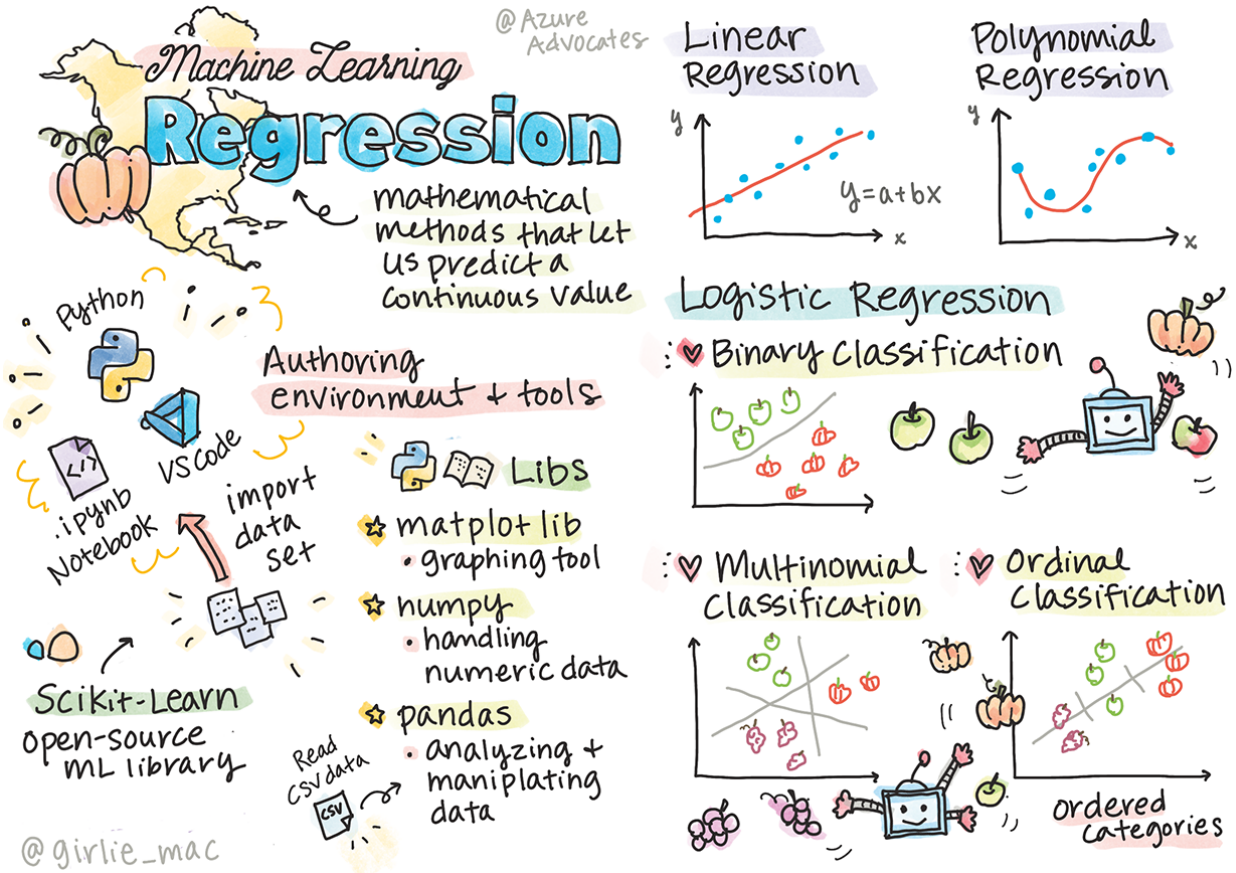
Introdution
Regression(Diabetes Data set)
Analyze and clean a Dataset (pumpkins dataset)
Correlation
Linear and Polynomial Regression using Scikit-Learn

Introdution

You'll learn about

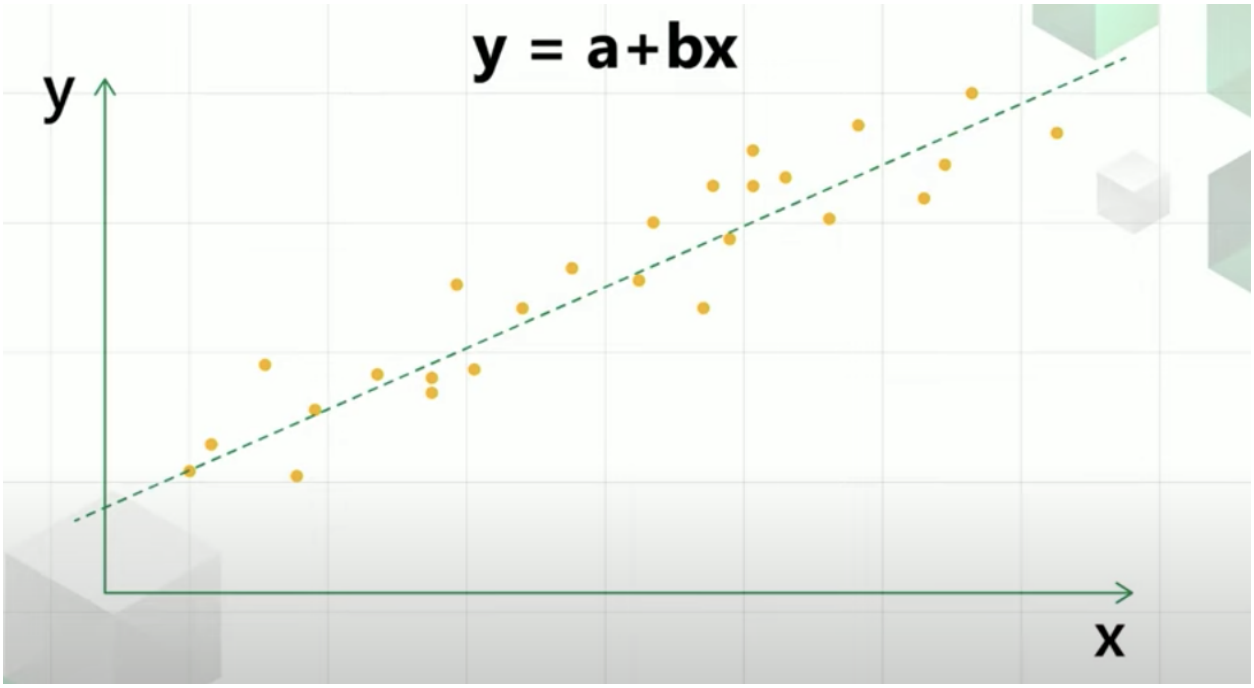
- Deciding whether AI is the right approach for your problem
- Collecting and preparing your data
- Training your model
- Evaluating your model
- Tuning the hyperparameters
- Testing the trained model in the real-world

Regression(Diabetes Data set)

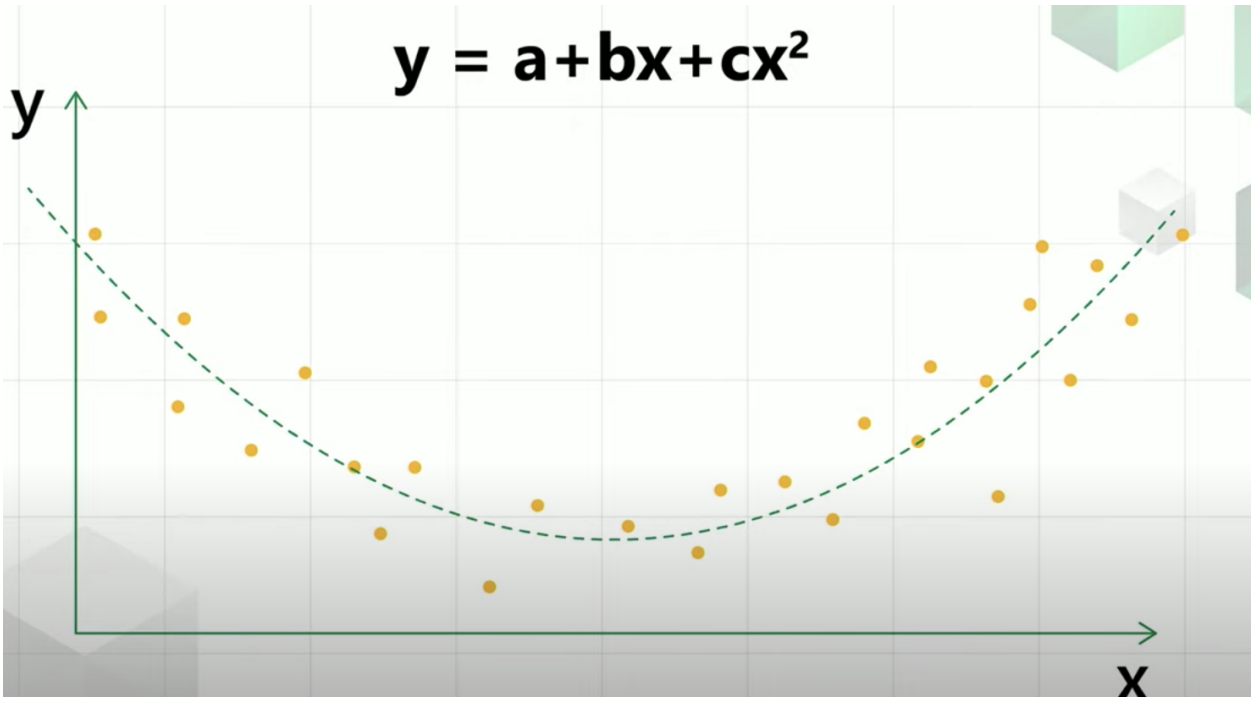


There is three type of Registration:

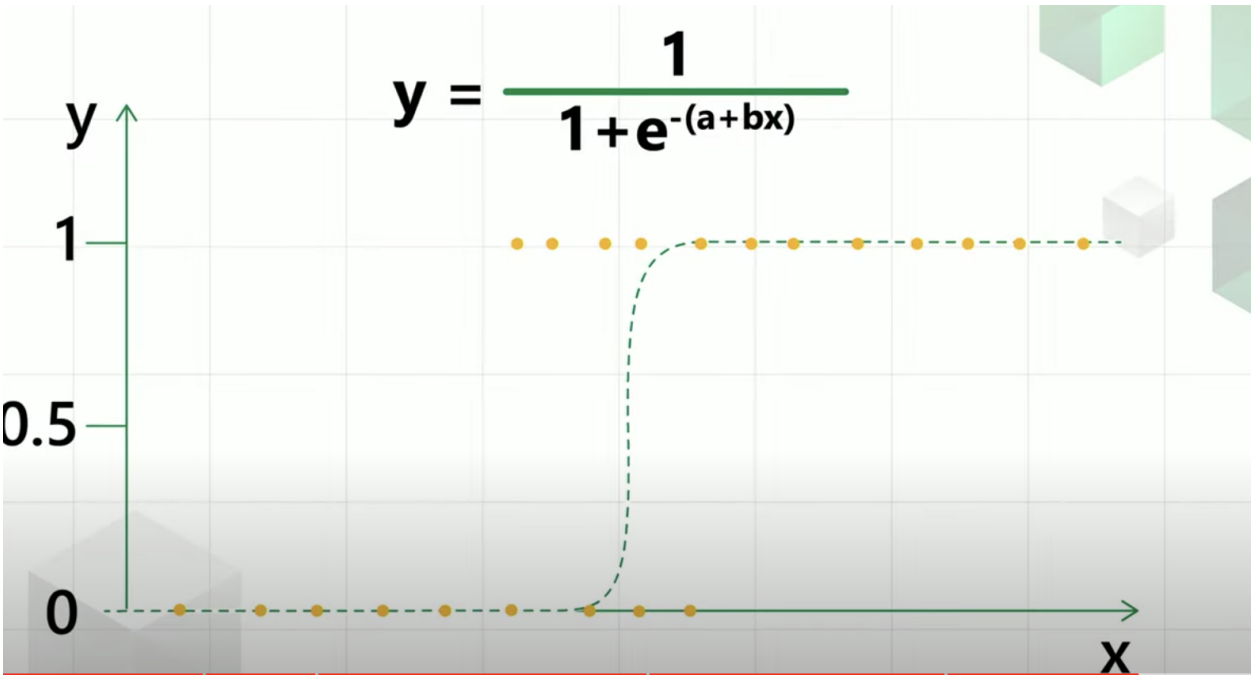
1- linear



2- polynomial



3- Logistics



give us probably of the y

for adding all file to my VS code local machine, should write on the terminal section on VS code

```
git chlone https://github.com/microsoft/ML-For-Beginners.git
```

for creating new directory on VS code write this on terminal section

```
python -m venv -h
python -m venv .venv
```

next is activate the environment

```
pip install pandas matplotlip numpy sci
kit-learn ipykernel
```

for check

```
pip list
```

to make sure we have numpy installed

```
import numpy as np
a = np.array ([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print(a[0])
```

Linear Regression

Diabetes Data

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html

Load and return the diabetes dataset (regression).

Samples total	442
Dimensionality	10
Features	real, -.2 < x < .2
Targets	integer 25 - 346

For loading the datasets

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
X, y = datasets.load_diabetes(return_X_y=True)

# Print the shape of the data snd the first row
print(X.shape)
print(X[0]))
```

(442, 10)
[0.03807591 0.05068012 0.06169621 0.02187239 -0.0442235 -0.03482076
-0.04340085 -0.00259226 0.01990749 -0.01764613]

we will set return X Y to True because we want a tuple containing the independent variable X sch as age, sex, BMI as well as depended variable Y which is the measure of the disease.

- we will keep the defaults for the other two parameters which means we will get the data as numpy arrays ad scikit-learn will automatically normalize them
- we have 442 data points with 10 attributes each

return_X_ybool, default=False
If True, returns (data, target) instead of a Bunch object. See below for more information about the data and target object.

at this point we just need a BMI that is in index 2

```
# Extract the column at Index 2
X = X [:, 2]
```

```
print(X.shape)

# Reshape to a 2D array
X = X.reshape((-1,1))
print(X.shape)
print(X)
```

Now we want to split out set to train data set and test data set

```
# split the data into training and testing data
from sklearn import model_selection
X_train,X_test, y_train, y_test = model_selection.train_test_spl
```

now create a Linear regression

```
# Create  a linear regression model and train it with our data
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

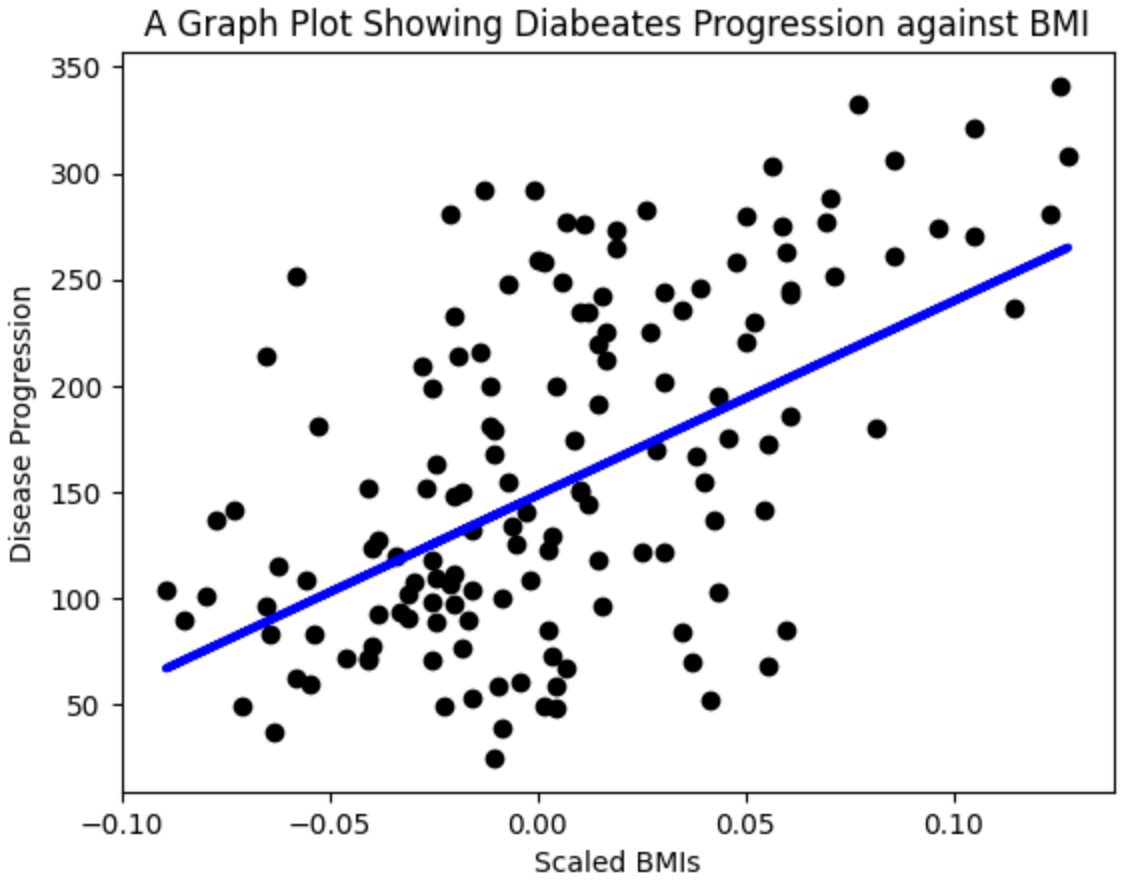
next we need to write a predict model

```
# Predict using our test data
y_prep = model.predict(X_test)
```

How good our prediction let use matplotlib to find out

```
# Create a scatter plot
plt.scatter(X_test,y_test, color ='black')

# Plot the prediction
plt.plot(X_test,y_pred, color = 'blue', linewidth = 3)

# Add labels and title
```

```
plt.xlabel('Scaled BMIs')
plt.ylabel('Disease Progression')
plt.title('A Graph Plot Showing Diabeates Progression against BM

# Draw the plot
plt.show()
```



Analyze and clean a Dataset (pumpkins dataset)

we use pumpkins data

https://github.com/microsoft/ML-For-Beginners/blob/main/2-Regression/data/US-pumpkins.csv

Import data

```
import pandas as pd
# Read the US pumpkins CSV file
pumpkins = pd.read_csv('/Users/hfakhrav/Downloads/ML-For-Beginne
```

Filter out any rows

```
# Filter out any rows that dont use pricw per bushel

pumpkins= pumpkins[pumpkins['Package'].str.contains('bushel', ca
pumpkins.head()
```

	City Name	Type	Package	Variety	Sub Variety	Grade	Date	Low Price	High Price	Mostly Low	...	Unit of Sale	Quality	Condition	Appearance	Storage	Crop	Repack	Trans Mode	Unnamed: 24	Unnamed: 25
70	BALTIMORE	NaN	1 1/9 bushel cartons	PIE TYPE	NaN	NaN	9/24/16	15.0	15.0	15.0	...	NaN	NaN	NaN	NaN	NaN	NaN	N	NaN	NaN	NaN
71	BALTIMORE	NaN	1 1/9 bushel cartons	PIE TYPE	NaN	NaN	9/24/16	18.0	18.0	18.0	...	NaN	NaN	NaN	NaN	NaN	NaN	N	NaN	NaN	NaN
72	BALTIMORE	NaN	1 1/9 bushel cartons	PIE TYPE	NaN	NaN	10/1/16	18.0	18.0	18.0	...	NaN	NaN	NaN	NaN	NaN	NaN	N	NaN	NaN	NaN
73	BALTIMORE	NaN	1 1/9 bushel cartons	PIE TYPE	NaN	NaN	10/1/16	17.0	17.0	17.0	...	NaN	NaN	NaN	NaN	NaN	NaN	N	NaN	NaN	NaN
74	BALTIMORE	NaN	1 1/9 bushel cartons	PIE TYPE	NaN	NaN	10/8/16	15.0	15.0	15.0	...	NaN	NaN	NaN	NaN	NaN	NaN	N	NaN	NaN	NaN

5 rows × 26 columns

we want to know many empty cell we have

```
# Count how many cell in each column are empty

pumpkins.isnull().sum()
```

```
City Name            0
Type               406
Package              0
Variety              0
Sub Variety        167
Grade              415
Date                 0
Low Price            0
High Price           0
Mostly Low          24
Mostly High         24
Origin               0
Origin District    396
Item Size          114
Color              145
Environment        415
Unit of Sale       404
Quality            415
Condition          415
Appearance         415
Storage            415
Crop               415
Repack               0
Trans Mode         415
Unnamed: 24        415
Unnamed: 25        391
dtype: int64
```

calculate the average price from the high and low price columns

```
# calculate the average price from the high and low price columı

price = (pumpkins['Low Price'] + pumpkins['High Price']) / 2
```

Get the month from the date column

```
# Get the month from the date column
month = pd.DatetimeIndex(pumpkins['Date']).month
```

Keep those columns that they are have 0 value and drop others

```
# Define the columns we want to keep

new_columns = ['Package','Month','Low Price','High Price', 'Date

# Drop all other columns

pumpkins = pumpkins.drop([c for c in pumpkins.columns if c not i

print(pumpkins.columns)
```

Index(['Package', 'Date', 'Low Price', 'High Price'], dtype='object')

Create new data frame

```
new_pumpkins = pd.DataFrame({
    'Month': month,
    'Package' : pumpkins['Package'],
    'Low Price' : pumpkins['Low Price'],
    'High Price' : pumpkins['High Price'],
    'Price': price
})
print(new_pumpkins.columns)
```

Index(['Month', 'Package', 'Low Price', 'High Price', 'Price'], dtype='object')

Month	Package	Low Price	High Price	Price
70	9 1 1/9 bushel cartons	15.0	15.0	15.0
71	9 1 1/9 bushel cartons	18.0	18.0	18.0
72	10 1 1/9 bushel cartons	18.0	18.0	18.0
73	10 1 1/9 bushel cartons	17.0	17.0	17.0
74	10 1 1/9 bushel cartons	15.0	15.0	15.0

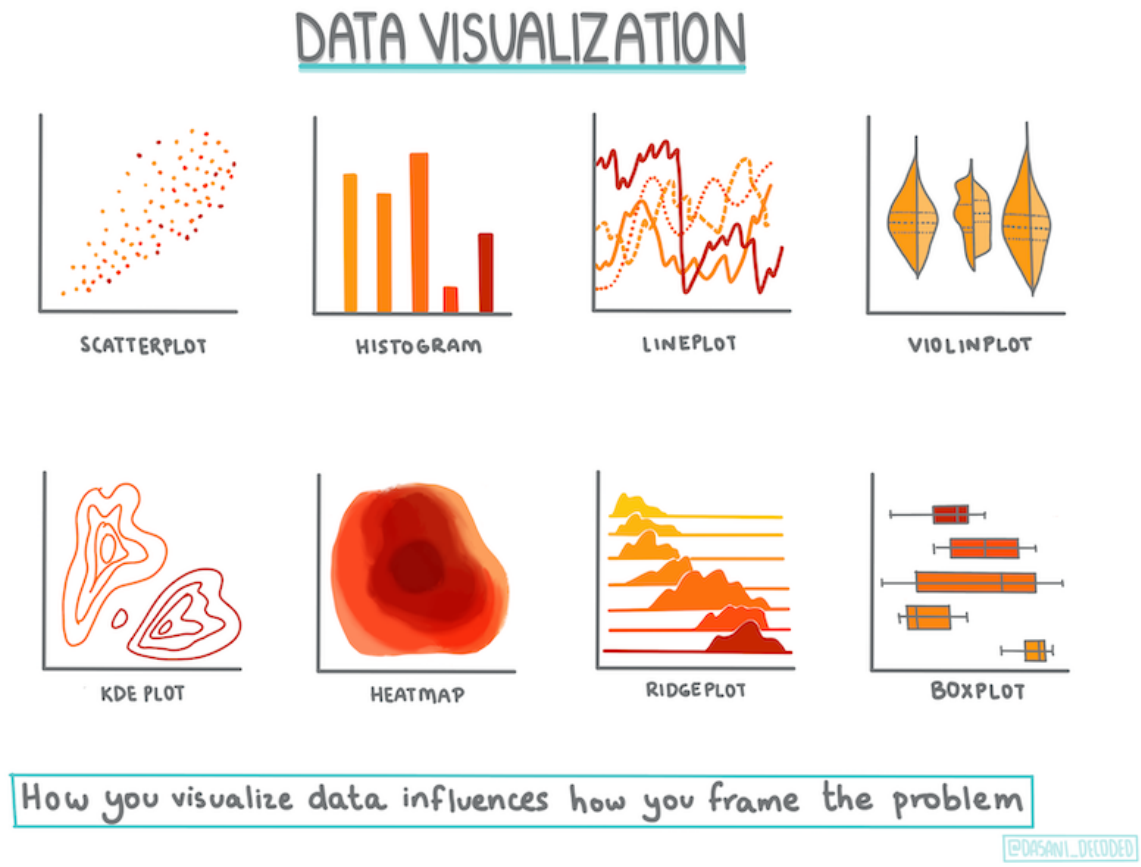
We still have bushel cartons problem there is two different size

we will normalize the average price

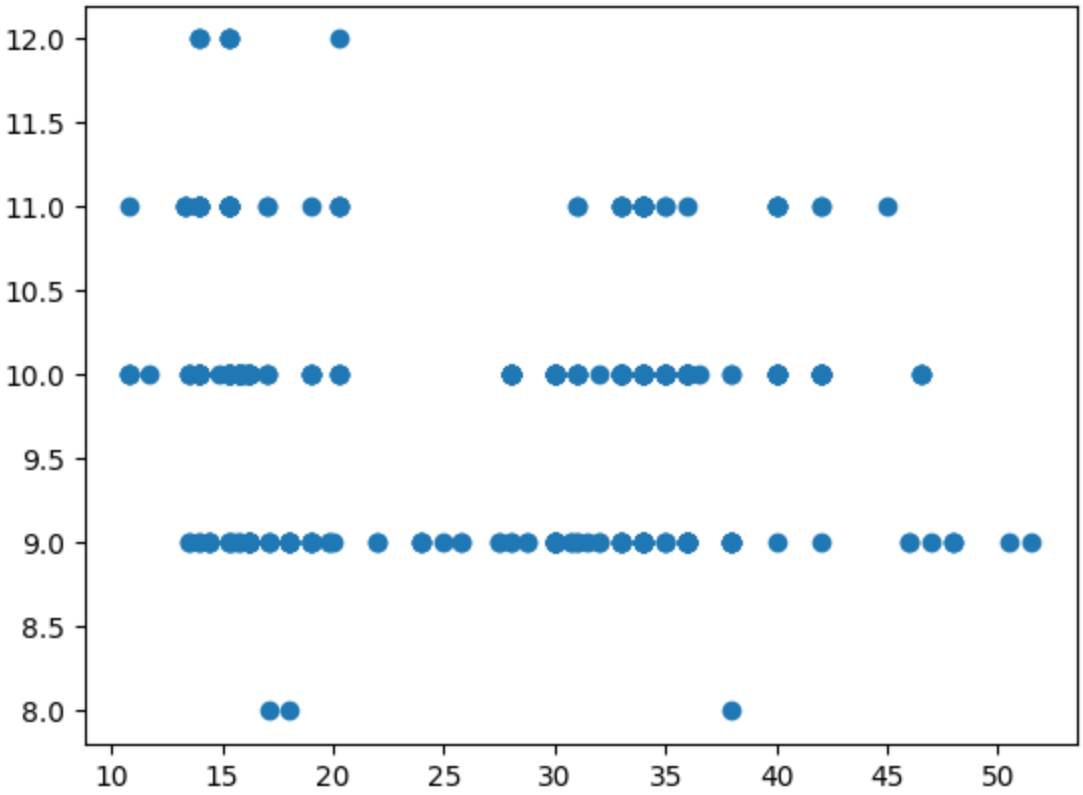
```
# Convert the price of all cells prices by 1 1/9 by dividing by

new_pumpkins.loc[new_pumpkins['Package'].str.contains('1 1/9'),

# Convert the price of all cells prices by  1/2 by dividing by
new_pumpkins.loc[new_pumpkins['Package'].str.contains('1/2'),'Pr

print(new_pumpkins.tail())
```

Month		Package	Low Price	High Price	Price
1738	9	1/2 bushel cartons	15.00	15.0	30.00
1739	9	1/2 bushel cartons	13.75	15.0	28.75
1740	9	1/2 bushel cartons	10.75	15.0	25.75
1741	9	1/2 bushel cartons	12.00	12.0	24.00
1742	9	1/2 bushel cartons	12.00	12.0	24.00

our question was which month are pumpkins are the cheapest



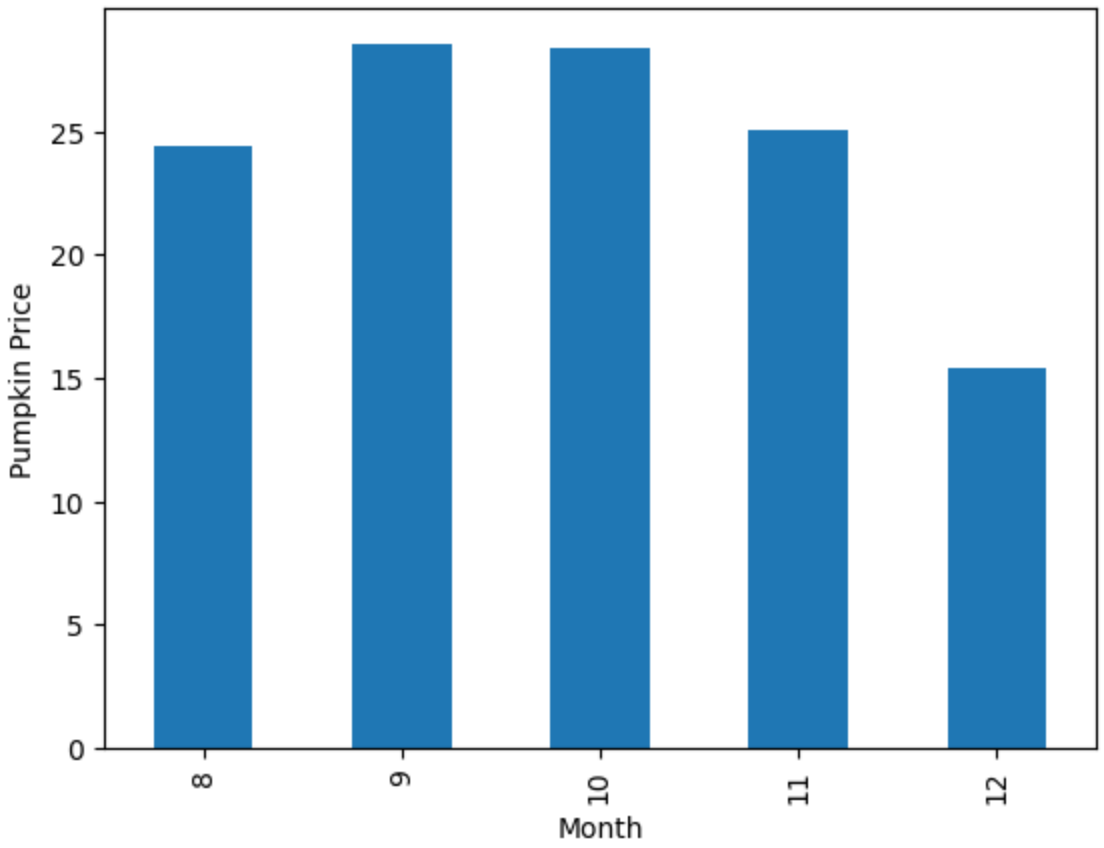

```
import matplotlib.pyplot as plt
# Get the values we want to plot
price = new_pumpkins.Price
month = new_pumpkins.Month

# Create a show a scatter plot of price vs month
plt.scatter(price, month)
plt.show()
```



Make it useful

To get charts to display useful data, you usually need to group the data somehow. Let's try creating a plot where the y axis shows the months and the data demonstrates the distribution of data.

```
# Group the pumpkins into a bar chart by month and price
new_pumpkins.groupby(['Month'])['Price'].mean().plot(kind = 'bar

# Add a label to the bar chart
plt.ylabel('Pumpkin Price')
```



A linear regression line

As you learned in Lesson 1, the goal of a linear regression exercise is to be able to plot a line to:

Show variable relationships. Show the relationship between variables

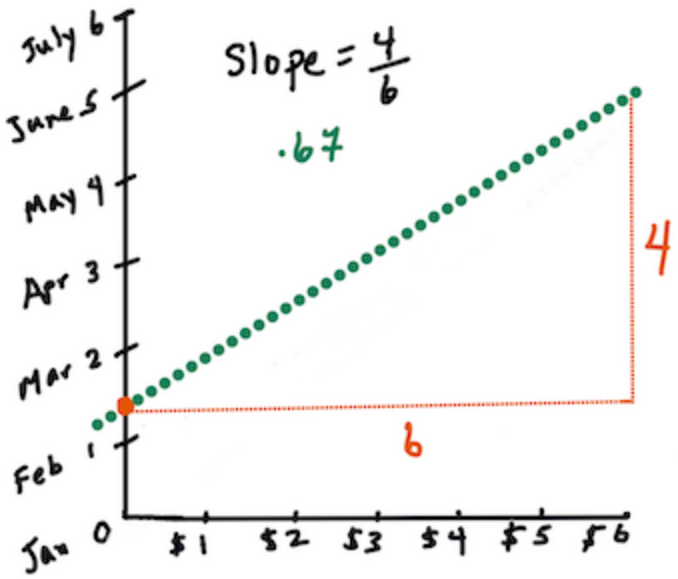
Make predictions. Make accurate predictions on where a new datapoint would fall in relationship to that line.

It is typical of Least-Squares Regression to draw this type of line. The term 'least-squares' means that all the datapoints surrounding the regression line are squared and then added up. Ideally, that final sum is as small as possible, because we want a low number of errors, or least-squares.

We do so since we want to model a line that has the least cumulative distance from all of our data points. We also square the terms before adding them since we are concerned with its magnitude rather than its direction.

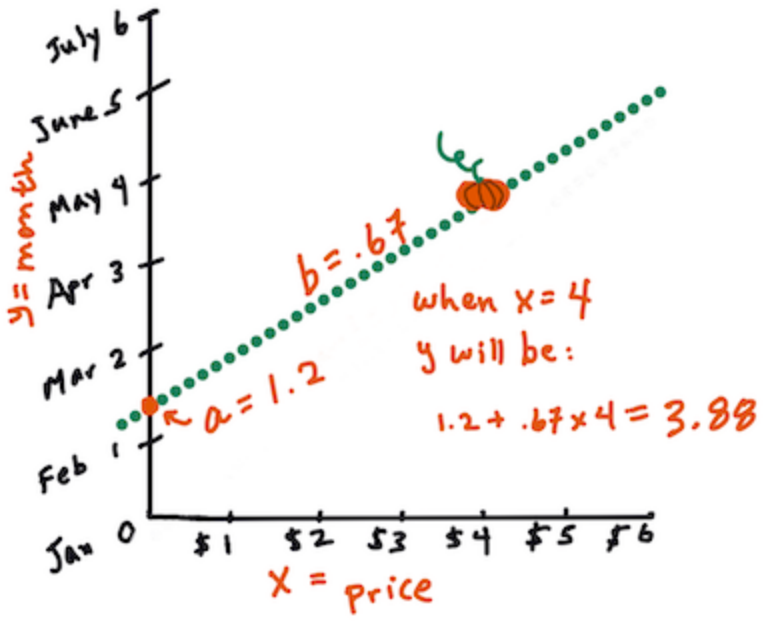
X is the 'explanatory variable'. Y is the 'dependent variable'. The slope of the line is b and a is the y-intercept, which refers to the value of Y when X = 0.

calculate the slope



First, calculate the slope b. Infographic by Jen Looper

In other words, and referring to our pumpkin data's original question: "predict the price of a pumpkin per bushel by month", X would refer to the price and Y would refer to the month of sale.



Calculate the value of Y. If you're paying around $4, it must be April!

Correlation

One more term to understand is the Correlation Coefficient between given X and Y variables. Using a scatterplot, you can quickly visualize this coefficient. A plot with datapoints scattered in a neat line have high correlation, but a plot with datapoints scattered everywhere between X and Y have a low correlation.

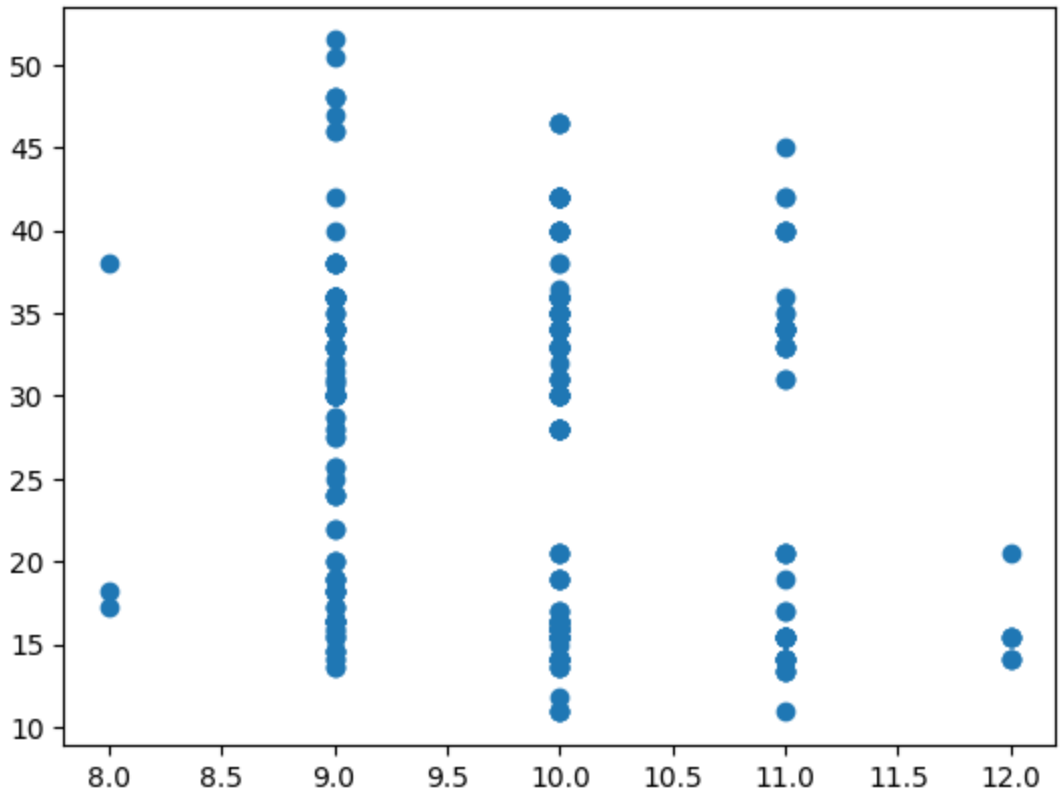
A good linear regression model will be one that has a high (nearer to 1 than 0) Correlation Coefficient using the Least-Squares Regression method with a line of regression.

```
pumpkins = pumpkins[pumpkins['Package'].str.contains('bushel', c

columns_to_select = ['Package', 'Variety', 'City Name', 'Low Pri
pumpkins = pumpkins.loc[:, columns_to_select]
```

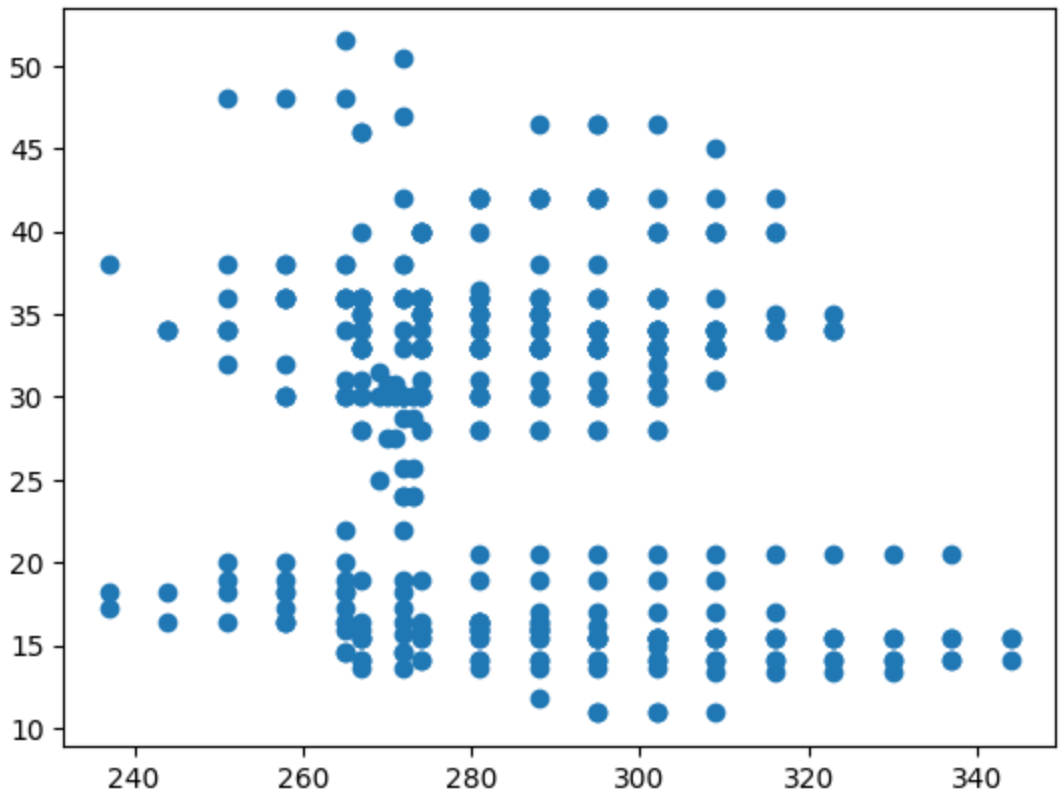
```
price = (pumpkins['Low Price'] + pumpkins['High Price']) / 2

month = pd.DatetimeIndex(pumpkins['Date']).month
day_of_year = pd.to_datetime(pumpkins['Date']).apply(lambda dt:

new_pumpkins = pd.DataFrame(
    {'Month': month,
     'DayOfYear' : day_of_year,
     'Variety': pumpkins['Variety'],
     'City': pumpkins['City Name'],
     'Package': pumpkins['Package'],
     'Low Price': pumpkins['Low Price'],
     'High Price': pumpkins['High Price'],
     'Price': price})

new_pumpkins.loc[new_pumpkins['Package'].str.contains('1 1/9'),
new_pumpkins.loc[new_pumpkins['Package'].str.contains('1/2'), 'P

new_pumpkins.head()
```

	Month	DayOfYear	Variety	City	Package	Low Price	High Price	Price
70	9	267	PIE TYPE	BALTIMORE	1 1/9 bushel cartons	15.0	15.0	13.636364
71	9	267	PIE TYPE	BALTIMORE	1 1/9 bushel cartons	18.0	18.0	16.363636
72	10	274	PIE TYPE	BALTIMORE	1 1/9 bushel cartons	18.0	18.0	16.363636
73	10	274	PIE TYPE	BALTIMORE	1 1/9 bushel cartons	17.0	17.0	15.454545
74	10	281	PIE TYPE	BALTIMORE	1 1/9 bushel cartons	15.0	15.0	13.636364

```
import matplotlib.pyplot as plt
plt.scatter('Month','Price',data=new_pumpkins)
```




```
plt.scatter('DayOfYear','Price',data=new_pumpkins)
```




```
# print the corelation between month and price
print(new_pumpkins['Month'].corr(new_pumpkins['Price']))
```

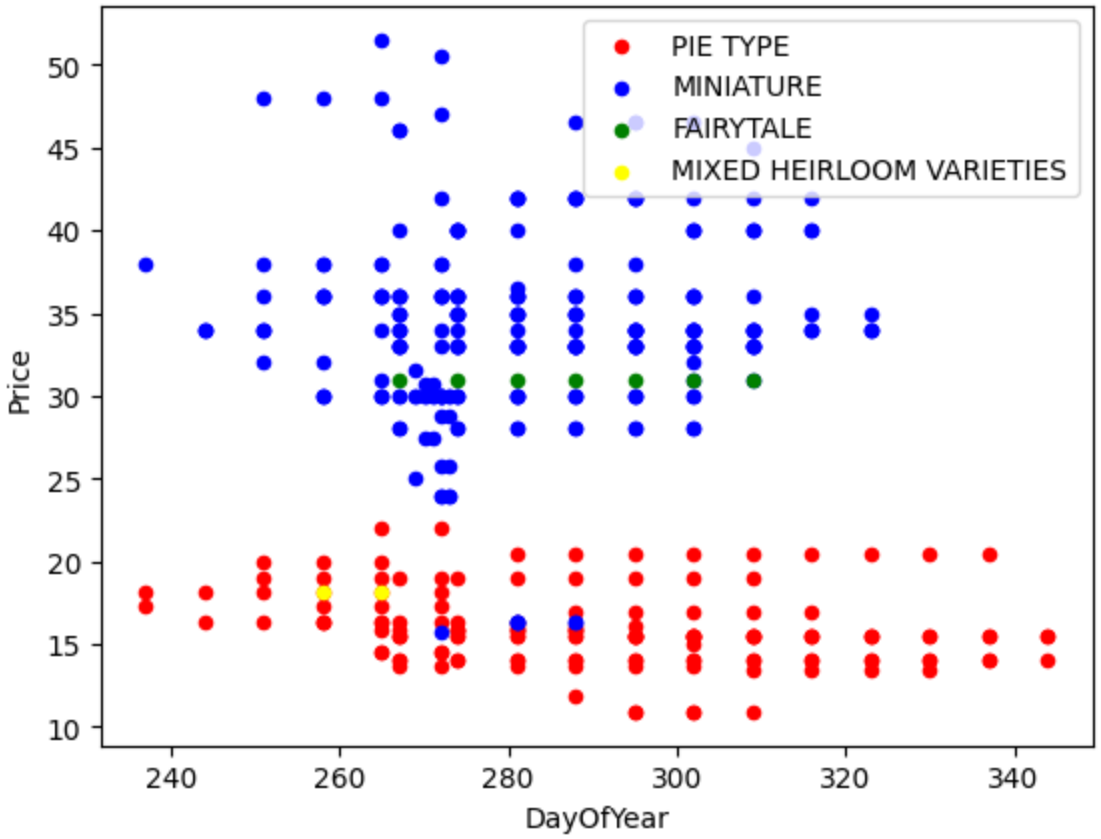
-0.14878293554077526

```
# print the corelation between Day of the year and price
print(new_pumpkins['DayOfYear'].corr(new_pumpkins['Price']))
```

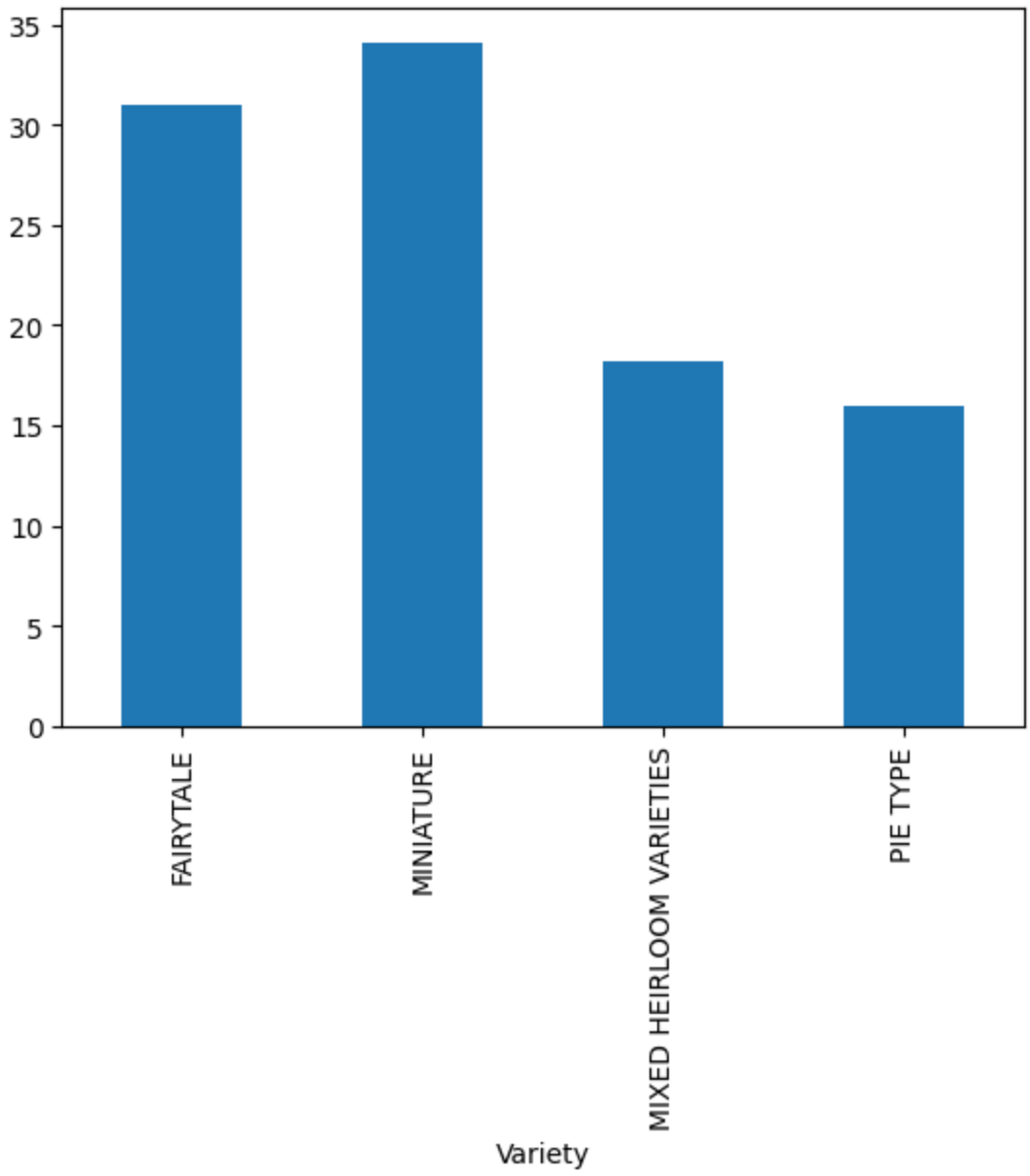
-0.1667332249274541

```
# Define the colors to use to plot the pumpkins
colors = ['red','blue','green','yellow']

#plot the price vs day of year for the pumkins, using a differe

ax = None
for i, var in enumerate(new_pumpkins['Variety'].unique()):
```

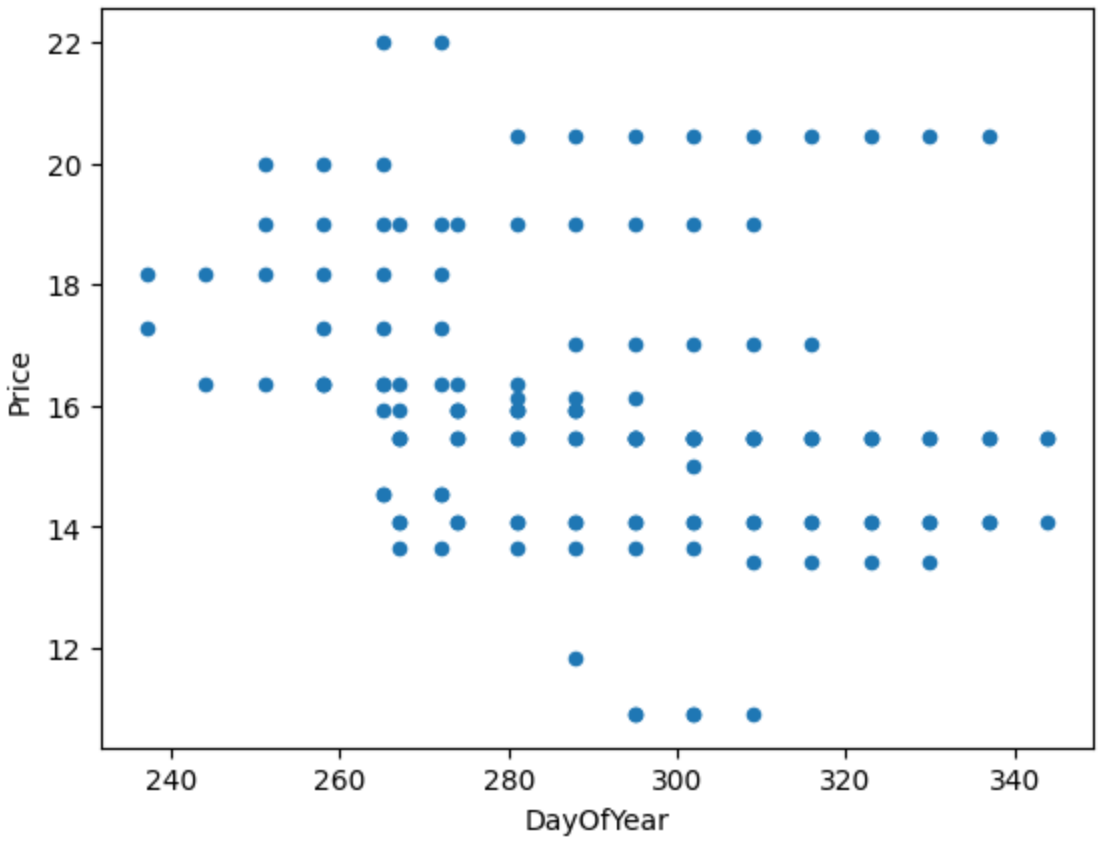
```
    df = new_pumpkins[new_pumpkins['Variety']== var]
    ax = df.plot.scatter('DayOfYear','Price', ax=ax, c = colors
```




```
new_pumpkins.groupby('Variety')['Price'].mean().plot(kind='bar'
```



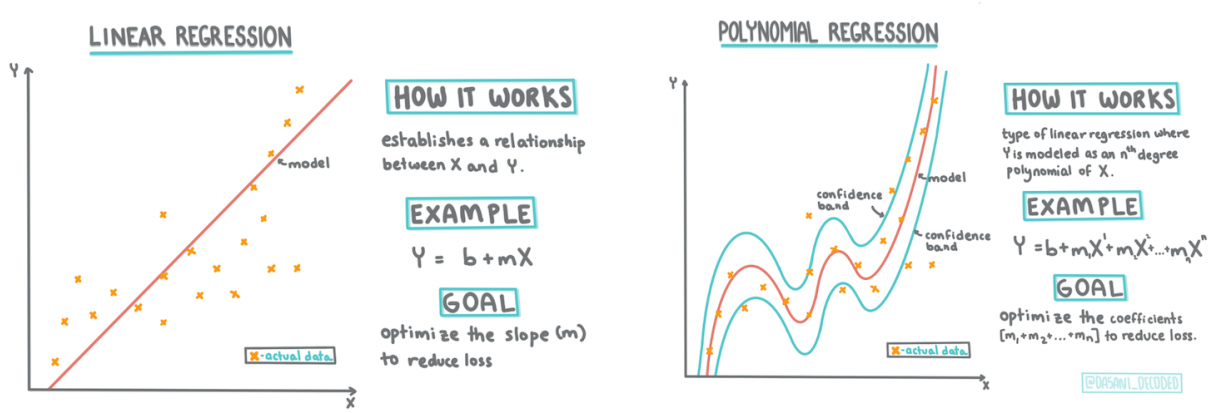

```
pie_pumpkins = new_pumpkins[new_pumpkins['Variety']== 'PIE TYPE
pie_pumpkins.plot.scatter('DayOfYear','Price')
```




```
# print the corelation between month and price
print(pie_pumpkins['Month'].corr(pie_pumpkins['Price']))
# print the corelation between Day of the year and price
print(pie_pumpkins['DayOfYear'].corr(pie_pumpkins['Price']))
```

-0.23841413206125747
-0.2669192282197318

Linear and Polynomial Regression using Scikit-Learn




```
# Get the day of year and price in separate arrays

x= pie_pumpkins['DayOfYear']
y = pie_pumpkins['Price']

# Print the shape
x.shape
```

(144,)

Remember that our goal with this model is to predict the price of pi type bushel of pupmlins given the day of the year. we need to reshape our input X because later we will be passing it to the fit function of linear regression and that expects a 2d array if we dont reshape it it will have shape 144 which is a 1D array

```
# we want to reshape it

# Get the day of year and price in separate arrays

x= pie_pumpkins['DayOfYear'].to_numpy().reshape(-1,1)
y = pie_pumpkins['Price']

# Print the shape
x.shape
```

(144, 1)

Next we need to split our data into test and train

```
# Split the data into training and testing data

X_train, X_test, y_train, y_test = train_test_split(X,y,test_siz
```

the train data set will be used to train the linear regression model and the test data set will be used to check the quality of our results we can then create our linear regression

```
# Create a Linear regression object
Lin_reg = LinearRegression()

# Train the model using out training data
Lin_reg.fit(X_train, y_train)
```

```
# Test the model using test data
pred = Lin_reg.predict(X_test)
pred
```

array([16.21096253, 16.73652536, 16.08833121, 15.35254325, 15.96569988, 16.73652536, 16.21096253, 16.36863138, 16.61389403, 15.84306855, 15.84306855, 15.72043723, 16.21096253, 15.5978059 , 16.45622519, 15.72043723, 15.5978059 , 16.08833121, 15.72043723, 16.36863138, 16.85915669, 16.21096253, 15.22991192, 16.21096253, 15.84306855, 16.45622519, 16.08833121, 15.84306855, 15.5978059])

if we print the pred variable we can see the price predicted for bushel of pie type pumpkins on different day of the year

How can we tell if these prediction are good we can calculate the mean squared error

```
# calculate the mean squared error

mse = np.sqrt(mean_squared_error(y_test,pred))

#print the mean squared erorr in an easy to read format

print(f' Mean error:{mse:3.3}({mse/np.mean(pred)*100:3.3}%)')
```

Mean error:2.77(17.2%)

That a pretty big error so our predictions wont be great

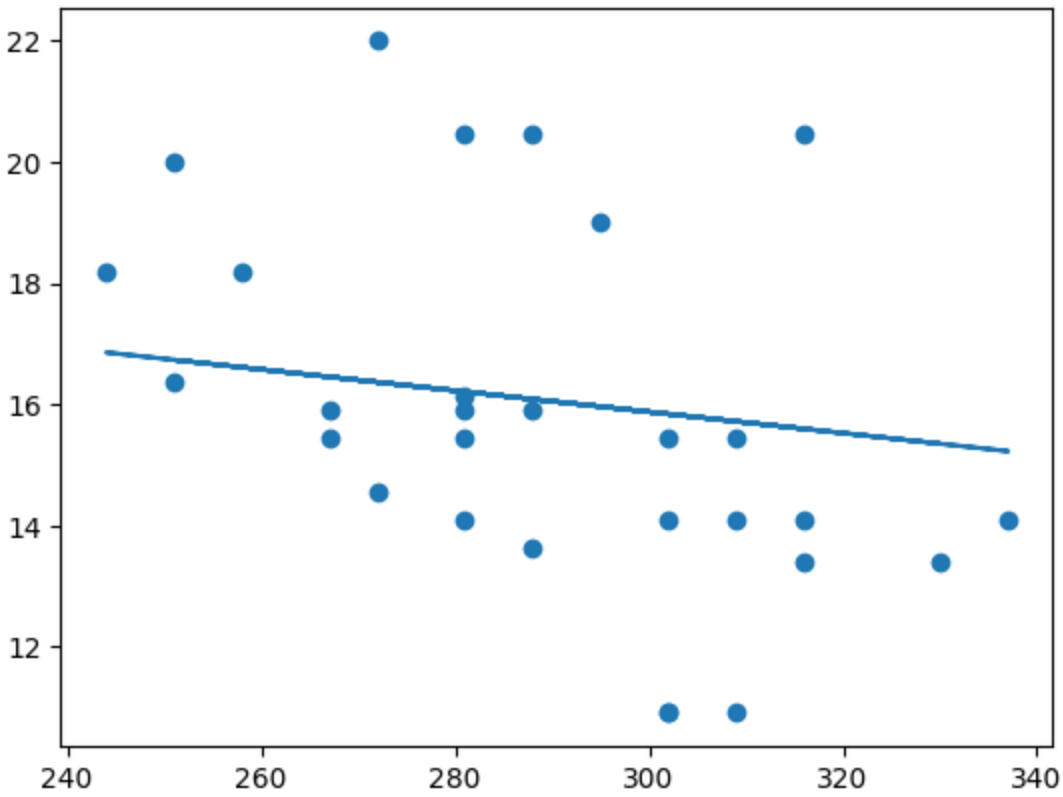
Another indicator of model quality is the coefficient of determination this value can be anywhere between 0 and 1 and the closer to one it is the better our model fits or the data a

```
# Calculate the coefficient of determination
score = Lin_reg.score(X_train, y_train)
print('Model determination: ', score)
```

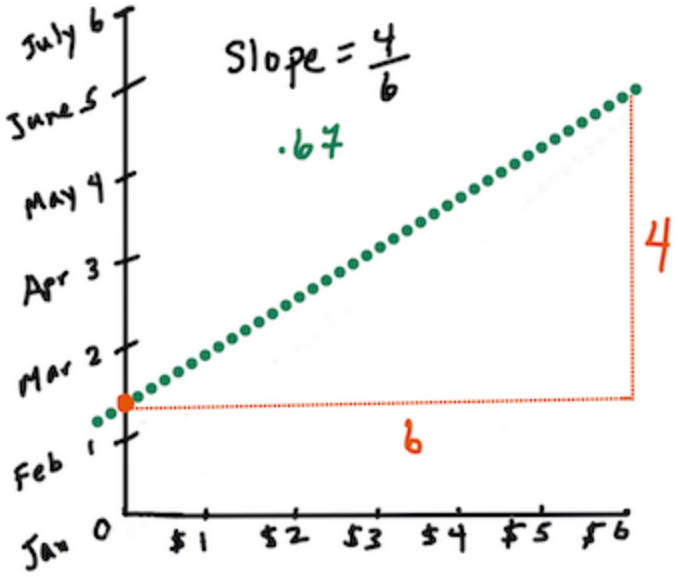
Model determination: 0.04460606335028361

which is very low

```
# Create a scatter plot using our test data
plt.scatter(X_test, y_test)

# Add a line to the plot with the presictions
plt.plot(X_test, pred)
```



we can calculate the slope




```
# Print the slope and the intercept
print(f'y={Lin_reg.coef_[0]}x + {Lin_reg.intercept_}')
```

y=-0.017518760953105x + 21.133734359909326

```
Lin_reg.predict([[265]])
```

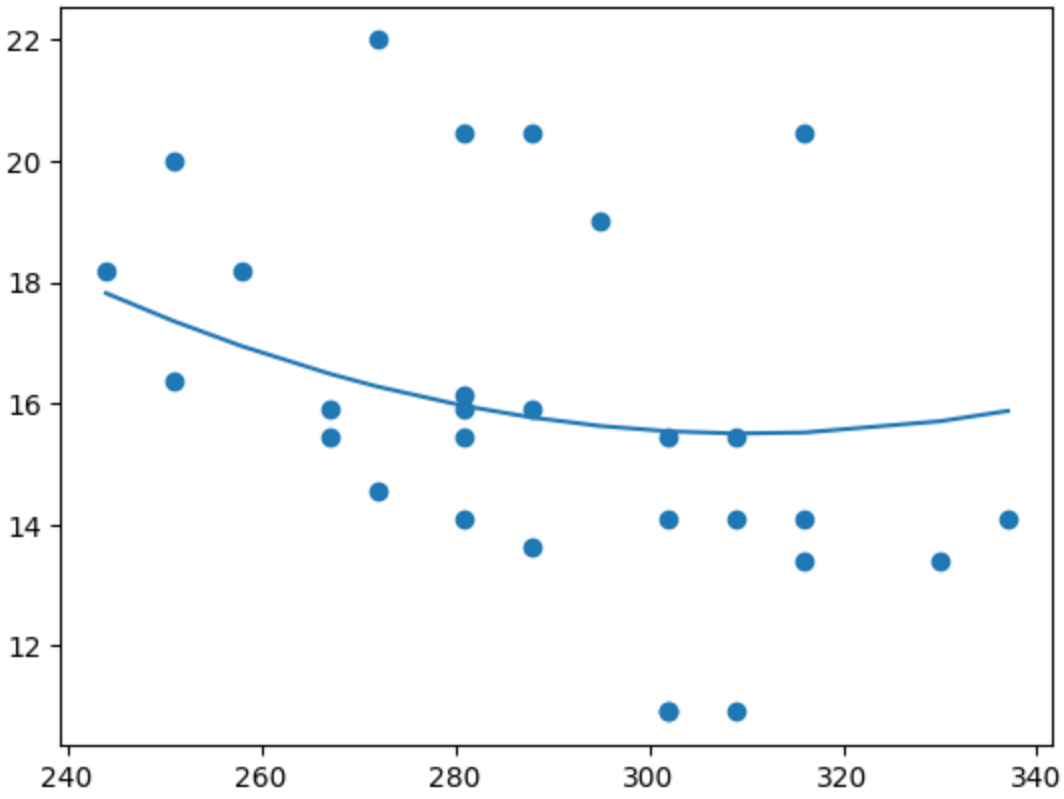
array([16.49126271])

lets use polynomial regression

```
# Build a polynomial regression pipeline
pipeline = make_pipeline(PolynomialFeatures(2), LinearRegressio

# Use the pipeline to build the model
pipeline.fit(X_train, y_train)

# Test the model with our test data
```

```
pred = pipeline.predict(X_test)

# Calculate and print the mean squared error
mse = np.sqrt(mean_squared_error(y_test,pred))
print(f'mena error: {mse:3.3}({mse/np.mean(pred)*100:3.3}%)')

# Plote the results
plt.scatter(X_test,y_test)
plt.plot(sorted(X_test),pipeline.predict(sorted(X_test)))
```

mena error: 2.73(17.0%)



let calculate coefficient of determination

```
# Score the model
score = pipeline.score(X_train, y_train)
print ('Model determination: ',)
```

Model determination: 0.07639977655280139