1. Using the example code we reviewed in class or by creating your own implementation, write a Python function to generate a random dataset in two dimensions, e.g. a collection of points {(xi, yi)} for i = 1, . . . , n. Specifically, use the numpy.random.rand() function to generate random xi values and cor- responding yi values then use the matplotlib library to plot a scatterplot of your dataset.

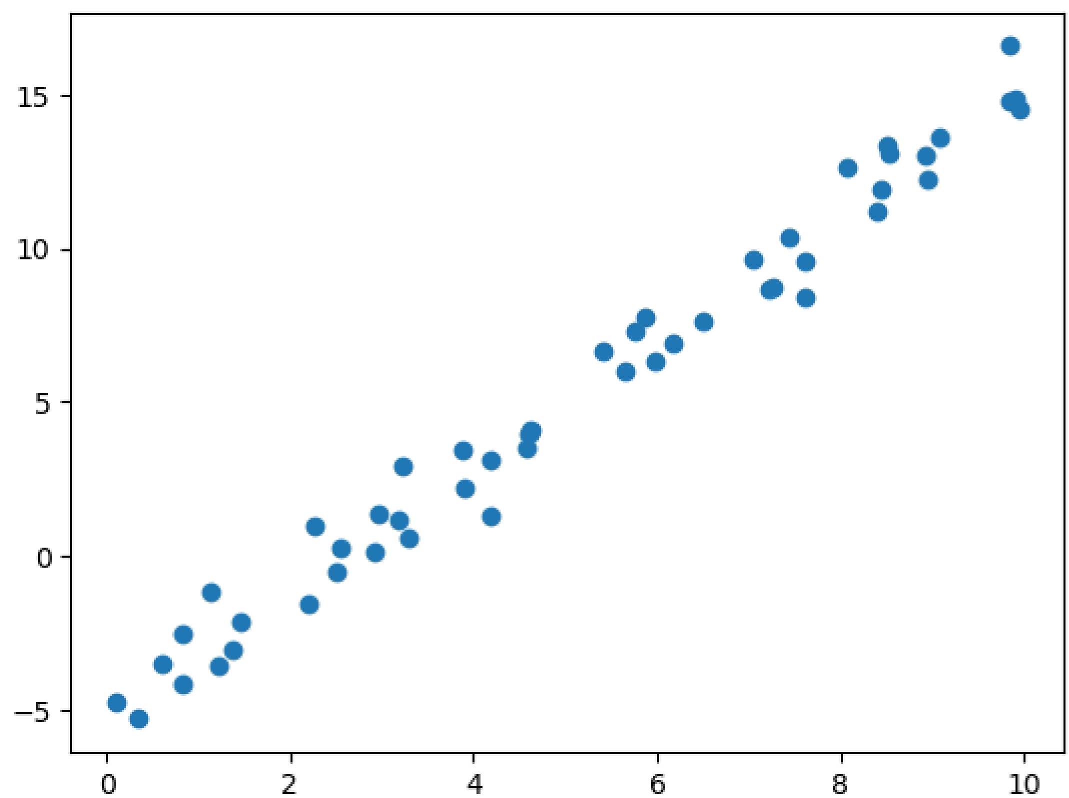
In [12]:
```
import pandas as pd
import numpy as np
```

In [13]:
```
#%matplotlib online
import matplotlib.pyplot as plt
```

In [14]:
```
import seaborn as sns#; sns.set
```

In [19]:
```
# y=numpy.random.rand(y0, y1, ..., yn);
# x=numpy.random.rand(x0, x1, ..., xn);
rng = np.random.RandomState(1)
x=10*np.random.rand(50)
y=2*x-5+0.9*rng.randn(50)
plt.scatter(x,y)
```

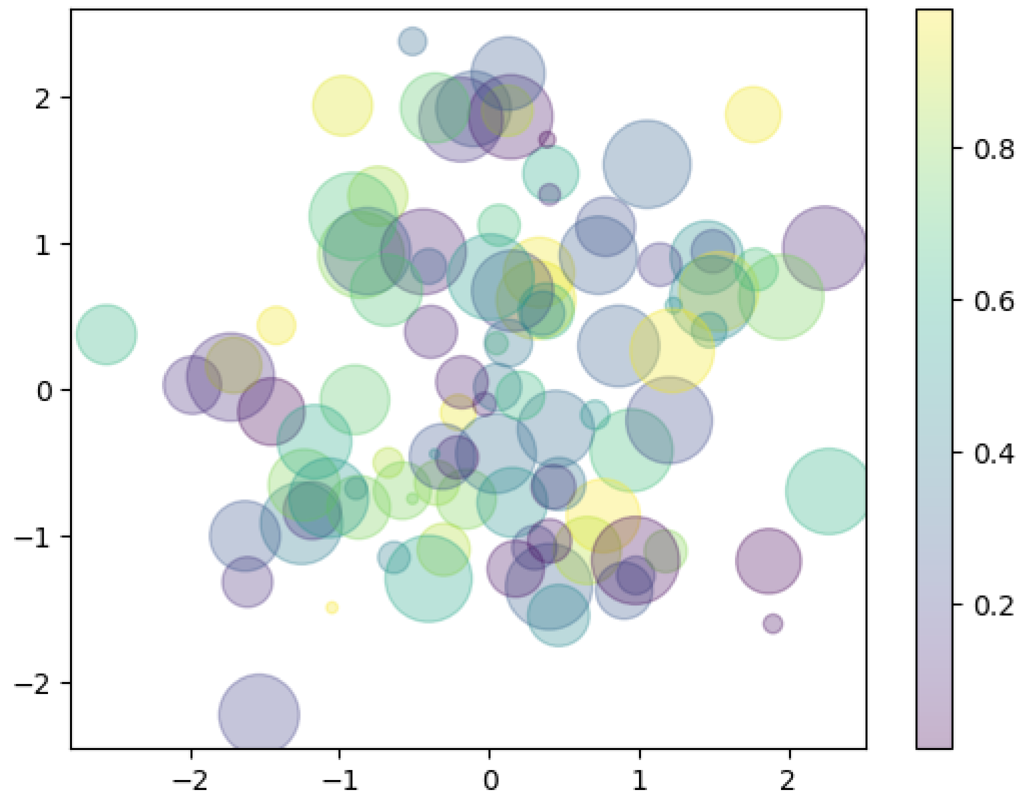
Out[19]:
```
<matplotlib.collections.PathCollection at 0x7fe963f50130>
```



It appears that in this example, x and y values have a positive relationship. Positive relationships have points that incline upwards to the right. As x values increase, y values increase. As x values decrease, y values decrease.

In [16]:
```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)
```

```
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
            cmap='viridis')
plt.colorbar();  # show color scale
```



2. Next fit the one dimensional linear model f(x)=β0 +β1x+ε and then plot your line f(x) = βˆ0 + βˆ1x overlaid upon the previous scatter plot of data.

In [4]:
```
# from sklearn.linear_model import LinearRegression
# model = LinearRegression(fit_intercept=True)
# model.fit(x[:,np.newaxix],y)
# xfit = np.linspace(0,10,1000)
# yfit = model.predict(xfit[:,np.newaxis])
# plt.scatter(x,y)
# plt.plot(xfit,yfit)
```

In [5]:
```
import numpy as np
from sklearn.linear_model
import LinearRegression
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
x
y = np.array([5, 20, 14, 32, 22, 38])
y
```

Out[5]:
```
array([ 5, 20, 14, 32, 22, 38])
```

In [6]:
```
model = LinearRegression()
model.fit(x, y)
```

Out[6]:
```
▾ LinearRegression
LinearRegression()
```

In [7]:
```
model = LinearRegression().fit(x, y)
```

In [10]:
```
r_sq = model.score(x, y)
print(f"coefficient of determination: {r_sq}")
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")
```

```
coefficient of determination: 0.715875613747954
intercept: 5.633333333333329
slope: [0.54]
```

In [11]:
```
new_model = LinearRegression().fit(x, y.reshape((-1, 1)))
print(f"intercept: {new_model.intercept_}")
y_pred = model.predict(x)
print(f"predicted response:\n{y_pred}")
y_pred = model.intercept_ + model.coef_ * x
print(f"predicted response:\n{y_pred}")
x_new = np.arange(5).reshape((-1, 1))
x_new
y_new = model.predict(x_new)
y_new
```

```
intercept: [5.63333333]
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
predicted response:
[[ 8.33333333]
 [13.73333333]
 [19.13333333]
 [24.53333333]
 [29.93333333]
 [35.33333333]]
```

Out[11]:
```
array([5.63333333, 6.17333333, 6.71333333, 7.25333333, 7.79333333])
```

In [ ]: