> The goal of data science is not to execute instead, to learn, and develop new business capabilities. Data Science is only useful when the data are used to answer the questions.

10 applications of DATA Science to reduce risk and quick processing in various domains is as below:

1. Fraud and Risk Detection
2. Healthcare
3. Internet Search
4. Targeted Advertising
5. Road Travel
6. Government
7. Website Recommendations
8. Advanced Image Recognition
9. Speech Recognition
10. Gaming

Outcome

> Submissions are evaluated using the Normalized Gini Coefficient.

> During scoring, observations are sorted from the largest to the smallest predictions. Predictions are only used for ordering observations; therefore, the relative magnitude of the predictions are not used during scoring. The scoring algorithm then compares the cumulative proportion of positive class observations to a theoretical uniform proportion.

> The Gini Coefficient ranges from approximately 0 for random guessing, to approximately 0.5 for a perfect score. The theoretical maximum for the discrete calculation is (1 - frac_pos) / 2.

> The Normalized Gini Coefficient adjusts the score by the theoretical maximum so that the maximum score is 1.

> The code to calculate Normalized Gini Coefficient in a number of different languages can be found in this forum thread.

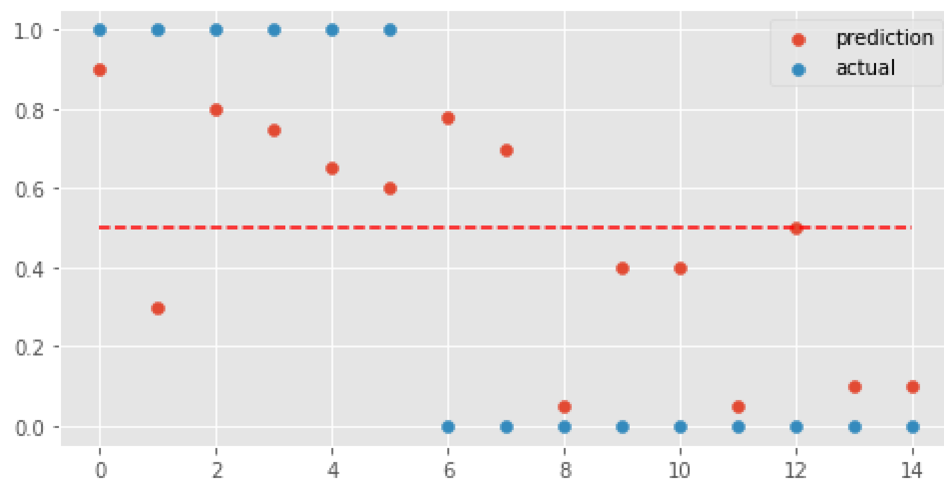
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate
import scipy.integrate

plt.style.use('ggplot')

predictions = [0.9, 0.3, 0.8, 0.75, 0.65, 0.6, 0.78, 0.7, 0.05, 0.4, 0.4, 0
actual = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [2]:
```
plt.figure(figsize = [8,4])
plt.scatter(x = range(len(predictions)), y = predictions, label='prediction
plt.scatter(x = range(len(actual)), y = actual,label = 'actual')
plt.plot(range(len(actual)),[0.5]*len(actual),'r--')
plt.legend()
```

Out[2]:
```
<matplotlib.legend.Legend at 0x1afc80b6b48>
```



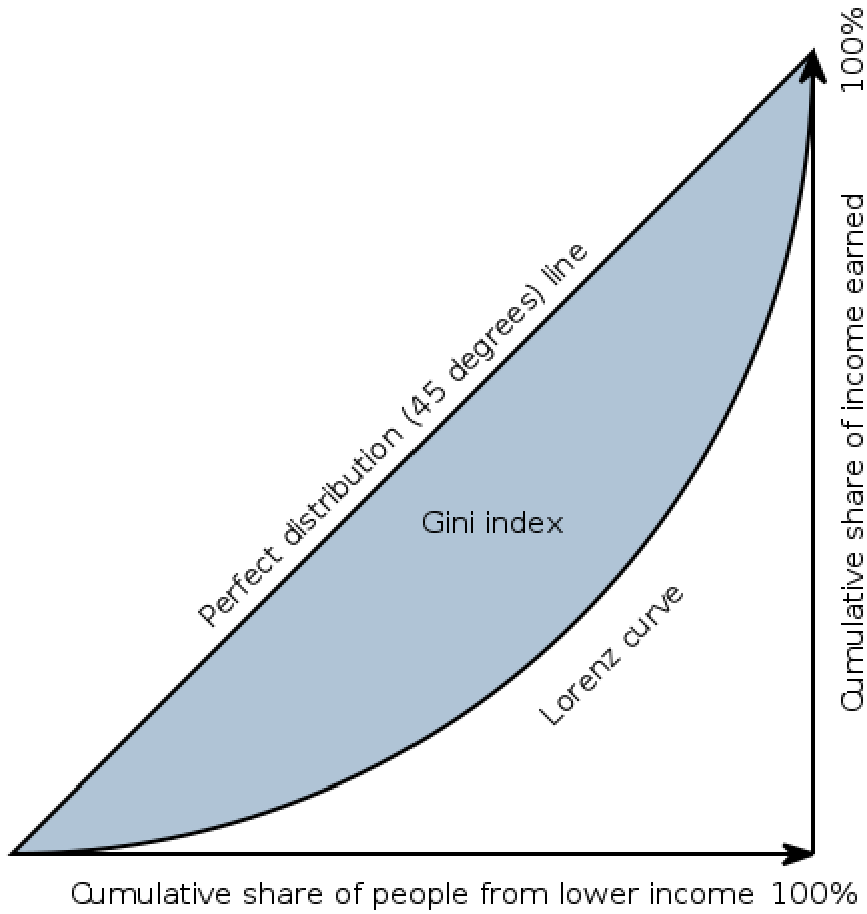
In [3]:
```
def gini(actual, pred):
    assert (len(actual) == len(pred))
    all = np.asarray(np.c_[actual, pred, np.arange(len(actual))], dtype=np.
    all = all[np.lexsort((all[:, 2], -1 * all[:, 1]))]
    totalLosses = all[:, 0].sum()
    giniSum = all[:, 0].cumsum().sum() / totalLosses

    giniSum -= (len(actual) + 1) / 2.
    return giniSum / len(actual)


def gini_normalized(actual, pred):
    return gini(actual, pred) / gini(actual, actual)


gini_predictions = gini(actual, predictions)
gini_max = gini(actual, actual)
ngini= gini_normalized(actual, predictions)
print('Gini: %.3f, Max. Gini: %.3f, Normalized Gini: %.3f' % (gini_predicti
```

```
Gini: 0.189, Max. Gini: 0.300, Normalized Gini: 0.630
```



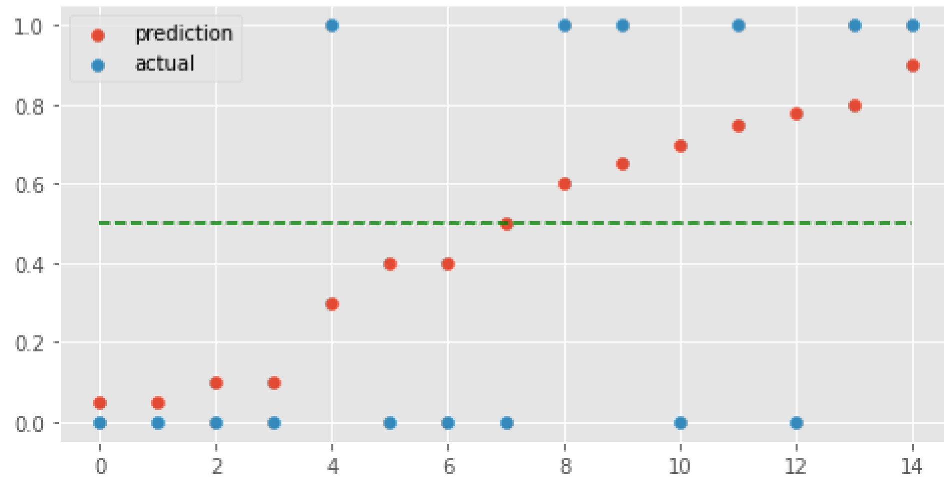
In [2]:

```
#data = zip(actual, predictions)
#sorted_data = sorted(data, key=lambda d: d[1])
#sorted_actual = [d[0] for d in sorted_data]
#print('Sorted Actual Values', sorted_actual)
```

Type *Markdown* and LaTeX: α^2

In [5]:
```
plt.figure(figsize = [8,4])
plt.scatter(x = range(len(predictions)), y = [d[1] for d in sorted_data], l
plt.scatter(x = range(len(actual)), y = [d[0] for d in sorted_data], label
plt.plot(range(len(actual)),[0.5]*len(actual),'g--')
plt.legend()
```

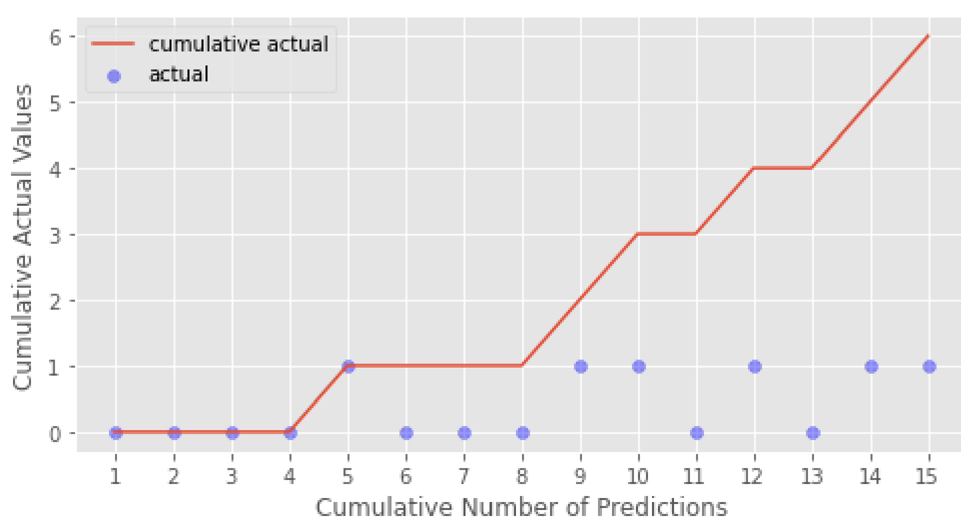
Out[5]: <matplotlib.legend.Legend at 0x1afc83df408>



In [6]:
```
# Sum up the actual values
cumulative_actual = np.cumsum(sorted_actual)
cumulative_index = np.arange(1, len(cumulative_actual)+1)


plt.figure(figsize = [8,4])
plt.plot(cumulative_index, cumulative_actual, label = 'cumulative actual')
plt.scatter(x = np.arange(1, len(cumulative_actual)+1), y = [d[0] for d in
plt.xlabel('Cumulative Number of Predictions')
plt.ylabel('Cumulative Actual Values')
plt.xticks(ticks = np.arange(1, len(cumulative_actual)+1))
plt.legend()
```

Out[6]: <matplotlib.legend.Legend at 0x1afd723fd88>



Type *Markdown* and LaTeX: α^2

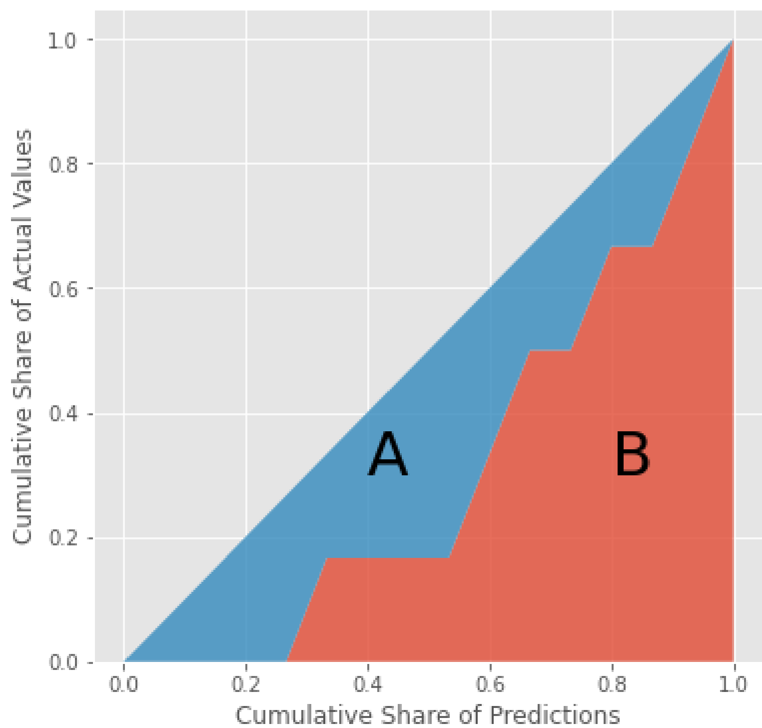
In [7]:
```
cumulative_actual_shares = cumulative_actual / sum(actual)
cumulative_index_shares = cumulative_index / len(predictions)

# Add (0, 0) to the plot
x_values = [0] + list(cumulative_index_shares)
y_values = [0] + list(cumulative_actual_shares)

# Display the 45° line stacked on top of the y values
diagonal = [x - y for (x, y) in zip(x_values, y_values)]

plt.figure(figsize = [6,6])
plt.stackplot(x_values, y_values, diagonal, alpha = .8)
plt.xlabel('Cumulative Share of Predictions')
plt.ylabel('Cumulative Share of Actual Values')

plt.text(x = 0.4, y = 0.3, s = 'A', fontsize = 30)
plt.text(x = 0.8, y = 0.3, s = 'B', fontsize = 30)
plt.show()
```



In [8]:
```
fy = scipy.interpolate.interp1d(x_values, y_values)
B, _ = scipy.integrate.quad(fy, 0, 1, points=x_values)
A = 0.5 - B
print(f'Size of A: {round(A,3)}')
```

```
Size of A: 0.189
```

Type *Markdown* and LaTeX: α^2

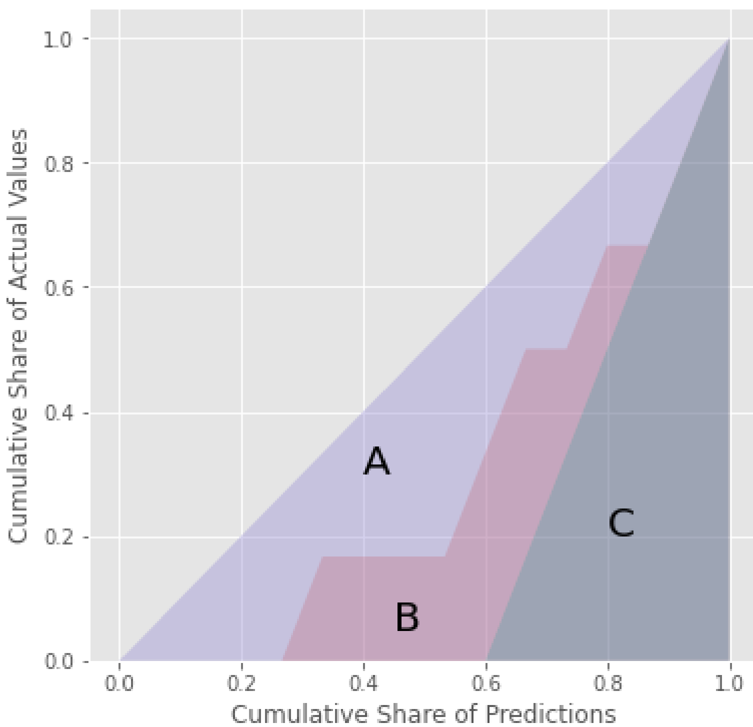
In [9]:
```
cumulative_actual_shares_perfect = np.cumsum(sorted(actual)) / sum(actual)
y_values_perfect = [0] + list(cumulative_actual_shares_perfect)

# Display the 45° line stacked on top of the y values
diagonal = [x - y for (x, y) in zip(x_values, y_values_perfect)]


plt.figure(figsize = [6,6])
plt.stackplot(x_values, y_values, alpha = .3)
plt.stackplot(x_values, y_values_perfect, diagonal, alpha = .4)
plt.xlabel('Cumulative Share of Predictions')
plt.ylabel('Cumulative Share of Actual Values')

plt.text(x = 0.4, y = 0.3, s = 'A', fontsize = 20)
plt.text(x = 0.45, y = 0.05, s = 'B', fontsize = 20)
plt.text(x = 0.8, y = 0.2, s = 'C', fontsize = 20)
plt.show()

# Integrate the the curve function
fy = scipy.interpolate.interp1d(x_values, y_values_perfect)
C, _ = scipy.integrate.quad(fy, 0, 1, points=x_values)
AB = 0.5 - C
print(f'A+B: {round(AB,3)}')
print(f'A/(A+B): {round(A/AB,3)}')
```




```
A+B: 0.3
A/(A+B): 0.63
```