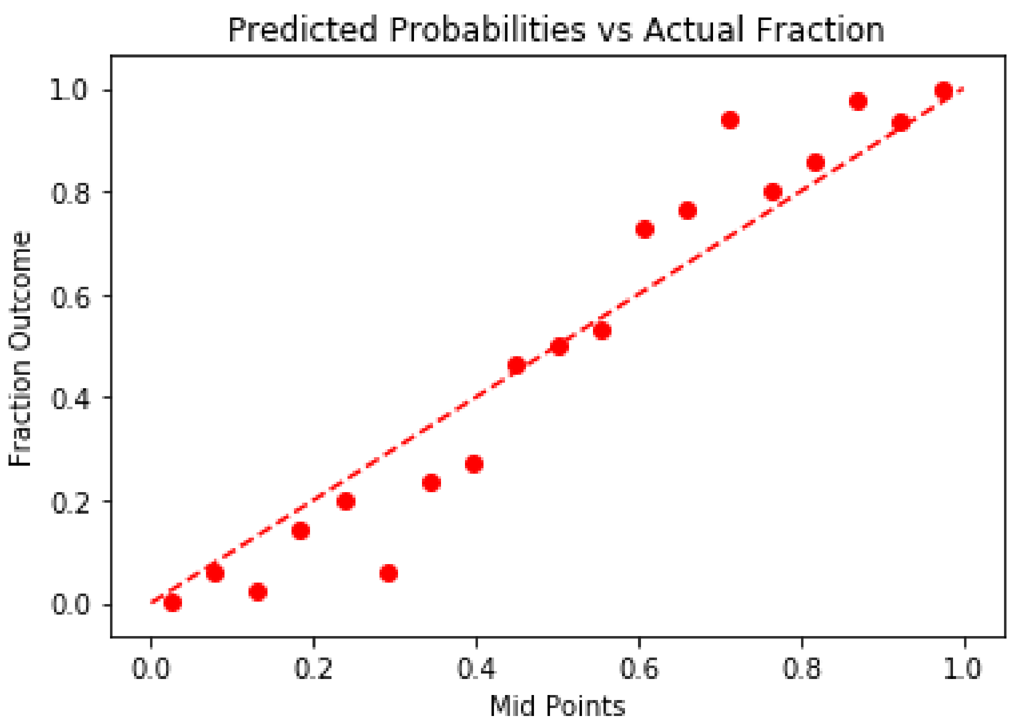
Notebook

April 20, 2019

Now make a scatterplot using `midpoints` as the x variable and `fraction_outcome` as the y variable. Draw a dashed line from `[0,0]` to `[1,1]` to mark the line y=x.

In [15]:
```
%matplotlib inline
import matplotlib.pyplot as plt

# scatterplot
x = midpoints
y = np.asarray(fraction_outcome)
plt.scatter(x,y, color = 'red')

# drawing a dashed line
xx= [0,1]
plt.plot(xx, color = 'red', linestyle='--' )

plt.title('Predicted Probabilities vs Actual Fraction')
plt.xlabel('Mid Points')
plt.ylabel('Fraction Outcome')

plt.show()
```



0.0.1 Question 5: adding error bars

If you did things correctly, it should look like fivethirtyeight has done "pretty" well with their forecasts: the actual fraction of wins tracks closely with the predicted number. But how do we decide what's "good enough"? Consider this example: I correctly predict that a coin is fair (e.g. that it has a 50% chance of heads, 50% chance of tails). But if I flip it 100 times, I can be pretty sure it won't come up heads exactly 50 times. The fact that it didn't come up heads exactly 50 times doesn't make my prediction incorrect.

To assess how reasonable the predictions are, I need to quantify the uncertainty in my estimate. It's reasonable to assume that within each bin, k, the observed number of wins, $Y_k \sim Bin(n_k, p_k)$, where n_k is the number of elections and p_k is the predicted win probability in bin k.

Classical results tell us that the obseved fraction of wins in bin k, $\hat{p} = \frac{Y_k}{n_k}$ has variance $\text{Var}(\hat{p}_k) = \frac{p_k(1-p_k)}{n_k} \approx \frac{\hat{p}_k(1-\hat{p}_k)}{n_k}$. The standard deviation of the Binomial proportion then is $\hat{\sigma}_k \approx \sqrt{\frac{\hat{p}_k(1-\hat{p}_k)}{n_k}}$.

If we use the normal approximation to generate a confidence interval, then the 95% interval has the form $\hat{p}_k \pm 1.96\hat{\sigma}_k$.

Create a new "aggregated" dataframe. This time, group `election_sub` by the `bin` and compute both the average of the `probwin_outcome` (`mean`) and the number of observations in each bin (`count`) using the `agg` function. Call this new data frame, `election_agg`.

```
In [16]: election_agg = election_sub.groupby('bin').agg({'probwin_outcome' : ['mean','count']})
         election_agg
```

```
Out[16]:                  probwin_outcome
                                    mean count
         bin
         ... Omitting 15 lines ...
         (0.842, 0.895]         0.976744    43
         (0.895, 0.947]         0.937500    32
         (0.947, 1.0]           0.998478   657
```

Use the mean and count columns of election_agg to create a new column of election_agg titled err, which stores $1.96 \times \hat{\sigma}_k$ in each bin k.

```
In [17]: # Using mean and count, to calculate variance for each bin
         election_agg = election_agg.assign(err = lambda x : 1.96* np.sqrt((x.iloc[:,0] *  (1- x.iloc[:
         election_agg.columns = ['mean', 'count', 'err']
         election_agg
```

```
Out[17]:                         mean  count       err
         bin
         (0.0, 0.0526]    0.001715    583  0.003359
         ... Omitting 14 lines ...
         (0.842, 0.895]   0.976744     43  0.045048
         (0.895, 0.947]   0.937500     32  0.083870
         (0.947, 1.0]     0.998478    657  0.002981
```

0.0.2 Question 7: understanding confidence intervals

Are the 95% confidence intervals generally larger or smaller for more confident predictions (e.g. the predictions closer to 0 or 1). What are the factors that determine the length of the confidence intervals?

The 95% confidence intervals are generally smaller for predictions that are closer to 0 or 1. The factors that determine the lengh of confidence intervals is the number of observations in each, and the probability of each individual outcome occuring. We see that in bins with a greater amount of observations have smaller confidence intervals. In the formula $\hat{\sigma}_k \approx \sqrt{\frac{\hat{p}_k(1-\hat{p}_k)}{n_k}}$, mathematically as count gets higher the variance gets lower. Generally, we are more "confident" in our predictions when there is a larger amount of trials/sample size hence the smaller confidence bands.

In [21]:
```python
# Input: a pandas dataframe with a numeric column named `probwin`
# Output: a pandas dataframe with the same columns, with an additional column named `absdiff`

def abs_diff(x):
    x['absdiff'] = x['probwin'].max() - x['probwin'].min()
    return x
```

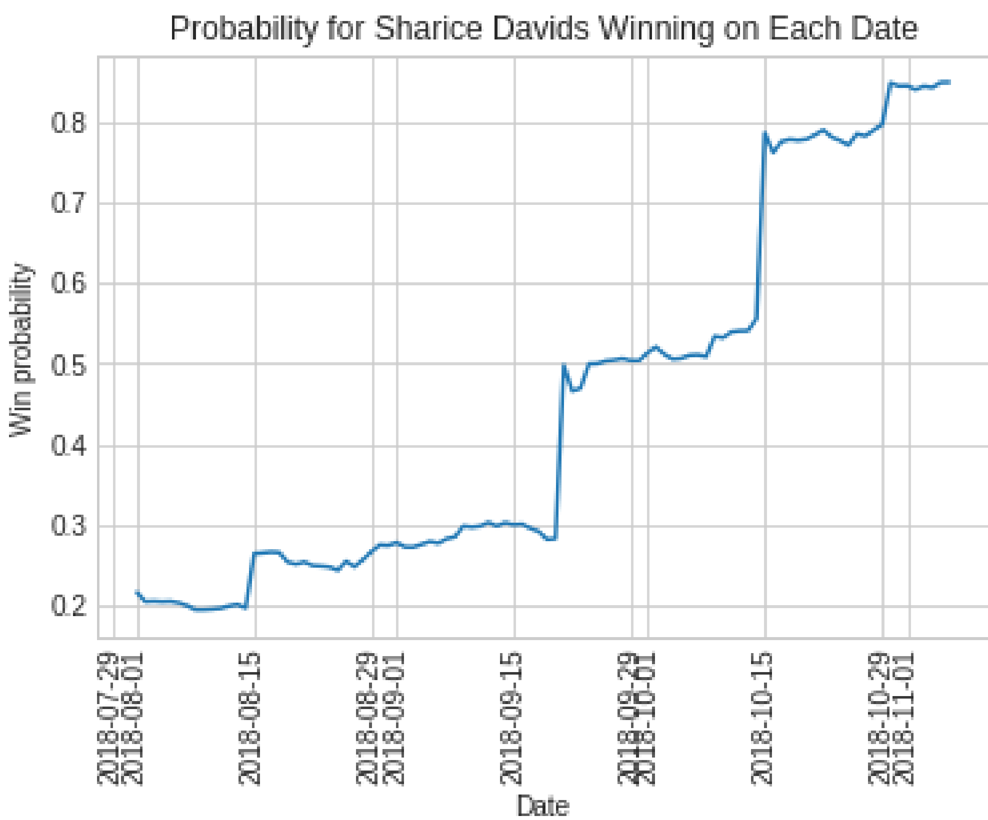
We can use this function to compute the difference between the maximum and minimum predicted with probabilities for every candidate. To do so, group election_sub by candidate and apply the function abs_diff. Find the index of the largest difference in diff_dataframe and store it in max_idx. Do this using np.nanargmax function. This function finds the *index* of the largest value, ignoring any missing values (nans).

```
In [22]: # Group candidates then apply abs diff function
         diff_dataframe = election_sub.groupby(['candidate']).apply(abs_diff)
         diff_dataframe

         # find index of largest value in column absdiff
         max_idx = np.nanargmax(diff_dataframe.absdiff)
         max_idx

Out[22]: 628
```

Did the candidate win or lose the election?
Sharice Davids won the election in her district.

Now create a lineplot with forecast date on the x-axis and the predicted win probability on the y-axis.

```
In [46]: plt.style.use('seaborn-whitegrid')
         # Need to fix X-axis for plot
         plt.plot(forecast_date, predicted_probs)
         plt.xticks(rotation=90)
         plt.title('Probability for Sharice Davids Winning on Each Date')

         plt.xlabel('Date')
         plt.ylabel('Win probability')
         plt.show()
```

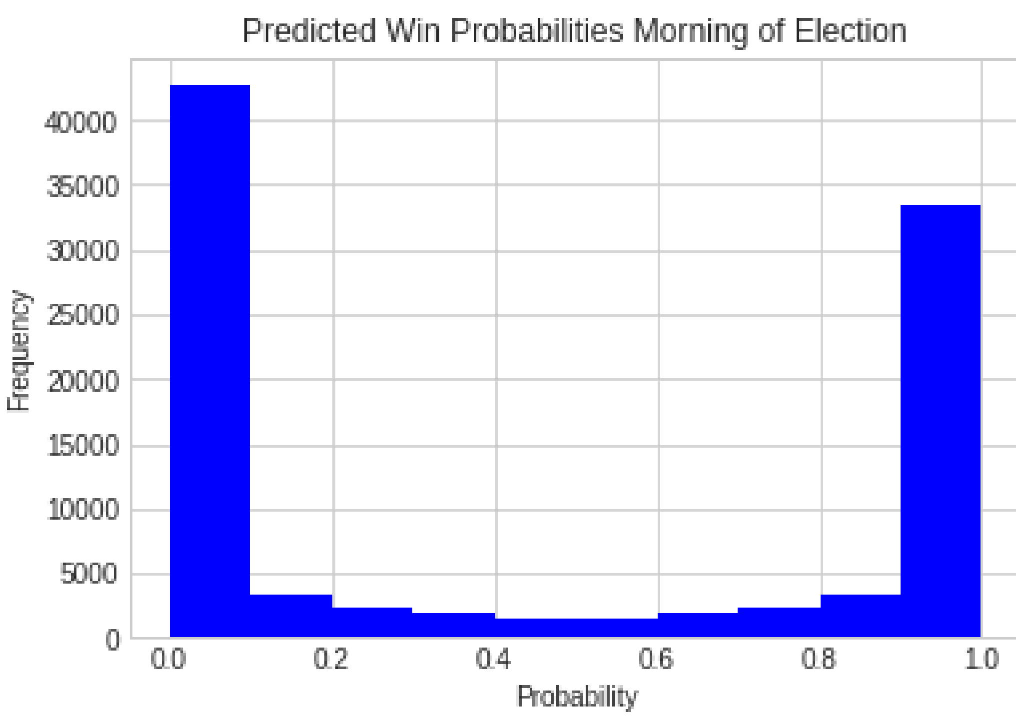


0.0.3 Question 10: prediction histograms

Make a histogram showing the predicted win probabilities on the morning of the election. Again, restrict yourself to only the `classic` predictions.

In [48]:
```
# histogram of prob1 for team1
plt.hist(all_election['probwin'], range=[0,1], color= 'blue')
plt.title("Predicted Win Probabilities Morning of Election")
plt.xlabel("Probability")
plt.ylabel("Frequency")


all_election['probwin']
```

Out[48]:
```
4         0.00032
5         0.99968
10        0.07366
... Omitting 55 lines ...
283141    0.74842
283142    0.25158
Name: probwin, Length: 94381, dtype: float64
```



Are most house elections easy to forecast or hard to forecast?

Yes, we can say that election are easy to forecast because there are a significant amount of candidates that had either a near a 0% chance of winning or near 100% chance of winning the election in there corresponding district.

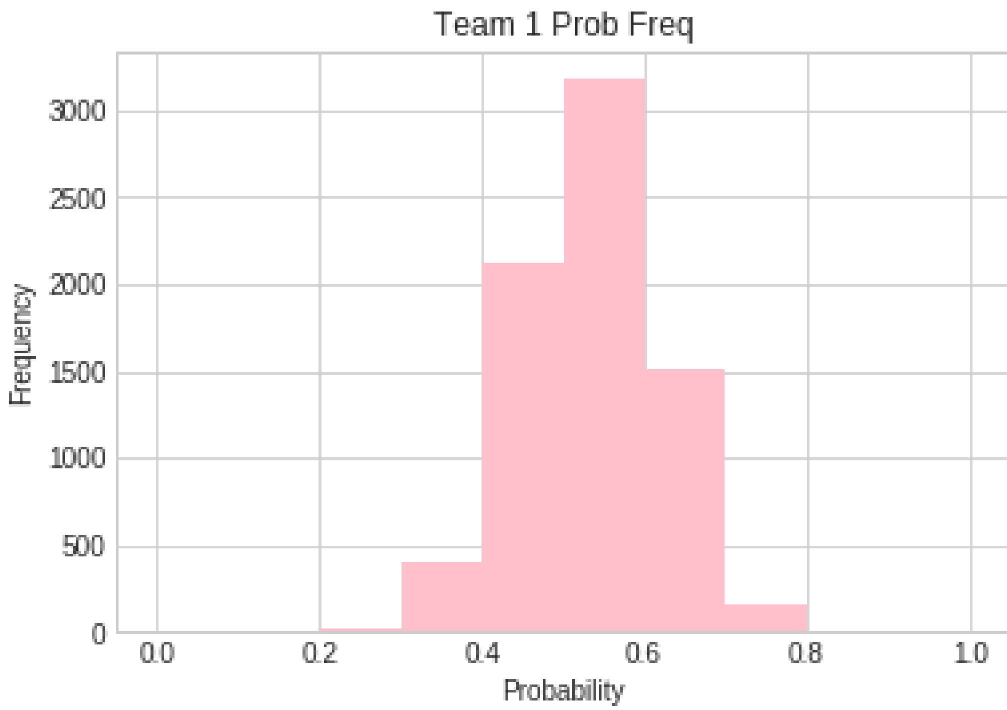
Create a pandas dataframe from the csv and print the first 10 rows.

```
In [31]: mlbgames = pd.read_csv("mlb_games.csv")

         mlbgames.iloc[0:9]
```

```
Out[31]:    season        date    team1    team2  dh     prob1  prob1_outcome  \
         0    2018  2018-10-28  Dodgers  Red Sox   0  0.483877            0.0
         1    2018  2018-10-27  Dodgers  Red Sox   0  0.508342            0.0
         ... Omitting 14 lines ...
         6  0.494789           0.0
         7  0.396971           1.0
         8  0.431984           1.0
```

In this dataframe `prob1` is the predicted win probability for `team1`. Make a histogram of `prob1`. Set the limits of the x-axis to `[0, 1]`

```
In [32]: import matplotlib.pyplot as plt


         # histogram of prob1 for team1
         plt.hist(mlbgames['prob1'], range=[0,1], color= 'pink')
         plt.title("Team 1 Prob Freq")
         plt.xlabel("Probability")
         plt.ylabel("Frequency")

Out[32]: Text(0, 0.5, 'Frequency')
```



0.0.4 Question 12

Find the most "surprising" baseball game outcome. To do so, select all of the entries for which `prob1_outcome` is 1 (i.e. `team1` won the game), and then look for the index of the row containing the smallest value of `prob1`. This will correspond to the game that was most suprising according to fivethirtyeights predictions. Find and print the row corresponding to this most surprising outcome.

```
In [33]: # Filtering rows with classic
         team1win_filter = mlbgames.prob1_outcome == 1

         #Applying filter where team 1 won game
         mlbfilter = mlbgames[team1win_filter]

         # Get row which has max value,
         # idx max returns index that has max value
         mlbfilter.loc[mlbfilter['prob1'].idxmin()]
```

```
Out[33]: season                 2018
         date             2018-08-25
         team1                Royals
         ... Omitting 3 lines ...
         prob2              0.712442
         prob2_outcome             0
         Name: 521, dtype: object
```

0.0.5 Question 13

Are the outcomes of baseball games generally easier or harder to predict than the outcomes of political elections? In a few sentences, comment on why this might be the case. What data is available for these predictions? What factors affect the outcomes of elections and baseball games? What makes an event like an election or a baseballgame "random"?

```
Building a predictive model for baseball games may be harder than political election ones because there
```

Create an analogous plot for empirical error bars with `bootstrap_election_agg`. Also draw a horizontal lines at 0 and 1.

SOLUTION HERE

Compare the two error bar plots and explain.
SOLUTION HERE