modeling

January 21, 2019

1 Data Science Final Project

1.1 Michael Alvin

1.2 Fall 2018

In [2937]:
```
import pandas as pd
import numpy as np
from math import sqrt
from scipy.optimize import curve_fit
from matplotlib import pyplot as plt
from pylab import rcParams
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from scipy import optimize
```

In [2938]:
```
def main():
#     df = pd.read_csv("dataset_spotify.csv")
#     create_bar_plots_rank(df)
#     remove_spotify_zero()
#     put_together()
#     parametrize_rank()
#     outliars_by_decade()
#     compare_ranks()
#     train_all()
#     feature_analysis()
    train_all()

    print("done")
```

2 Progress Report

Below is code written for the progress report.

3 model_dataset(df)

model_dataset(df) fits the dataset with linear regression.

```
In [2939]: def model_dataset(df):
               df['old_popularity'] = 101 - df['rank']

               train, test = train_test_split(df, test_size=0.2)
               features_cols=['years_released','old_popularity']
               X=train[features_cols]
               y=train['spotify_popularity']

               lin_reg = Ridge(alpha=1.0)
               lin_reg.fit(X, y)
               y_pred=lin_reg.predict(X)

           #     X_test=test[features_cols]
           #     y_pred=lin_reg.predict(X_test)

               train['predicted_spotify_popularity'] = y_pred
               train['difference'] = abs(train['spotify_popularity'] - train['predicted_spotif
               train['ratio'] = train['spotify_popularity']/train['predicted_spotify_popularit

               print(train['difference'].describe())
               print(lin_reg.coef_)

           #     test['predicted_spotify_popularity'] = y_pred
           #     test['difference'] = abs(test['spotify_popularity'] - test['predicted_spotify_
           #     test['ratio'] = test['spotify_popularity']/test['predicted_spotify_popularity

           #     print(test['difference'].describe())

               df.to_csv("predictions.csv", index=False)
```

4 create_bar_plots_rank(df)

create_bar_plots_rank(df) simply creates a few bar graphs:

- x: rank, y: median spotify_popularity for that rank

```
In [2940]: def create_bar_plots_rank(df):
               df = df.groupby(["Peak Position"])["Peak Position", "Spotify Popularity"].median

               x = df["Peak Position"]
               y = df["Spotify Popularity"]

               mask1 = (y < 35)
               mask2 = (y >= 35) & (y < 40)
```

```
    mask3 = (y >= 40) & (y < 45)
    mask4 = (y >= 45) & (y < 50)
    mask5 = (y >= 50) & (y < 55)
    mask6 = (y >= 55)

    plt.bar(x[mask1], y[mask1], color = 'red')
    plt.bar(x[mask2], y[mask2], color = 'orange')
    plt.bar(x[mask3], y[mask3], color = 'yellow')
    plt.bar(x[mask4], y[mask4], color = 'green')
    plt.bar(x[mask5], y[mask5], color = 'blue')
    plt.bar(x[mask6], y[mask6], color = 'violet')

    label('Peak Position', 'Spotify Popularity Median', 'Median Current Popularity R

    plt.savefig('median_current_popularity_rank.png')
```

5 create_bar_plots_year(df)

create_bar_plots_year(df) simply creates a few bar graphs:

- x: year, y: median spotify_popularity for that year

```
In [2941]: def create_bar_plots_year(df):
    df = df.groupby(["Billboard Year"])["Billboard Year", "Spotify Popularity"].medi

    x = df["Billboard Year"]
    y = df["Spotify Popularity"]

    mask1 = (y < 20)
    mask2 = (y >= 20) & (y < 30)
    mask3 = (y >= 30) & (y < 40)
    mask4 = (y >= 40) & (y < 50)
    mask5 = (y >= 50) & (y < 60)
    mask6 = (y >= 60)

    plt.bar(x[mask1], y[mask1], color = 'red')
    plt.bar(x[mask2], y[mask2], color = 'orange')
    plt.bar(x[mask3], y[mask3], color = 'yellow')
    plt.bar(x[mask4], y[mask4], color = 'green')
    plt.bar(x[mask5], y[mask5], color = 'blue')
    plt.bar(x[mask6], y[mask6], color = 'violet')

    label('Year of Release', 'Spotify Popularity Median', 'Median Current Popularity

    plt.savefig('median_current_popularity_year.png')
```

6 create_line_plots_rank(df)

create_line_plots_rank(df) simply creates a few more line graphs:

- x: rank, y: rank of median spotify_popularity for that rank compared to other ranks
- x: rank, y: rank of median youtube_viewcount for that rank compared to other ranks

```
In [2942]: def create_line_plots_rank(df):
               df = df.groupby(["rank"])["rank", "spotify_popularity", "youtube_viewcount"].mea

               df['spotify_popularity_rank'] = df['spotify_popularity'].rank(ascending=True)
               df['spotify_popularity_rank'] = df['spotify_popularity_rank'].astype(int)

               df['youtube_popularity_rank'] = df['youtube_viewcount'].rank(ascending=True)
               df['youtube_popularity_rank'] = df['youtube_popularity_rank'].astype(int)

           #     print(df)

               spotify_ranking = df.plot.scatter(x='rank', y='spotify_popularity_rank', rot=90
               plt.savefig('spotify_ranking_plot_rankly (median).png')

           #     youtube_ranking = df.plot.scatter(x='rank', y='youtube_popularity_rank', rot=
           #     plt.savefig('youtube_ranking_plot_rankly.png')
```

7 create_line_plots_year(df)

create_line_plots_year(df) simply creates a few more line graphs:

- x: year, y: rank of median spotify_popularity for that year compared to other years
- x: year, y: rank of median youtube_viewcount for that year compared to other years

```
In [2943]: def create_line_plots_year(df):
               df = df.groupby(["year"])["year", "spotify_popularity", "youtube_viewcount"].mea

               df['spotify_popularity_rank'] = df['spotify_popularity'].rank(ascending=True)
               df['spotify_popularity_rank'] = df['spotify_popularity_rank'].astype(int)

               df['youtube_popularity_rank'] = df['youtube_viewcount'].rank(ascending=True)
               df['youtube_popularity_rank'] = df['youtube_popularity_rank'].astype(int)

           #     print(df)

               spotify_ranking = df.plot.scatter(x='year', y='spotify_popularity_rank', rot=90
               plt.savefig('spotify_ranking_plot_yearly (median).png')

           #     youtube_ranking = df.plot.scatter(x='year', y='youtube_popularity_rank', rot=
           #     plt.savefig('youtube_ranking_plot_yearly.png')
```

```
#     spotify_youtube = df.plot.scatter(x='spotify_popularity_rank', y='youtube_pop
#     plt.xlabel('spotify_popularity_rank', fontsize=16)
#     plt.ylabel('youtube_popularity_rank', fontsize=16)
#     plt.savefig('youtube_spotify_ranks.png')
```

8 clean_data(df)

clean_data(df) simply adds column headers on to the complete dataset for readability and replaces 0 values for with imputed mean of that year.

```
In [2944]: def clean_data(df):
               df = pd.read_csv("complete_dataset.csv", header=None)
               df.columns = ["year", "rank", "title", "artist",
                             "spotify_popularity", "youtube_viewcount"]

               df["spotify_popularity"] = df["spotify_popularity"].replace(0, np.nan) # replac
               df["youtube_viewcount"] = df["youtube_viewcount"].replace(0, np.nan) # replace

               df = df.dropna()

               years_df = pd.DataFrame(columns=["years_released"])
               years_df["years_released"] = 2017 - df["year"]
               df.insert(loc=4, column='years_released', value=years_df["years_released"])

               df.to_csv("dataset.csv", index=False)
```

9 Final Report

Below is code written for the final report.

10 inspect(df)

inspect(df) simply inspects dataset for statistics that may be interesting. Here we are exploring statistics of spotify popularity of all songs to see the distribution, because the scores are not uniformly distributed [0, 100]. Furthermore, we wanted to see the year counts of the song, how many songs produced in specific years.

```
In [2945]: def inspect(df):
               fig, ax = plt.subplots()
               df["spotify_popularity"].value_counts().sort_index().plot(ax=ax, kind='bar')
               print(df["spotify_popularity"].describe())

               df = df.groupby([df['year']]).agg({'count'})
               print(df)
```

11 remove_duplicates_merge()

remove_duplicates_merge() removes duplicates from the originial Billboard weekly data, removing by both 'Song' and 'Artist'. From the original list of approximately 260000 songs, about 26000 songs remained. This means each song appeared on the Billboard list approximately 10 times in its lifetime. First, we dropped and keeped first to find the 'Billboard Year' as we assumed its first appearance on Billboard as its year of release, so a song who first appeared on Billboard 05/07/1980 is assumed to have its year of release as 1980. Then, we dropped and keeped last to find the 'Peak Position' and 'Weeks on Chart', because the last row would have the maximum peak position and weeks on chart of each song. We merged the keep='first' and keep='last' and the result is our base dataset.

```
In [2946]: def remove_duplicates_merge():
               df = pd.read_csv("billboard_hot_1960_2017.csv")
               df_left = df.drop_duplicates(subset=['Song', 'Artist'], keep='first')
               df_right = df.drop_duplicates(subset=['Song', 'Artist'], keep='last')

               df_left = df_left.drop(['Last Week', 'Peak Position', 'Weeks on Chart'], axis=1)
               df_right = df_right.drop(['Rank', 'Last Week', 'Date'], axis=1)

               df_new = df_left.merge(df_right, how='left', left_on=['Song','Artist'], right_on

               df_new["Billboard Year"] = pd.to_datetime(df_new["Date"], format='%Y-%m-%d')
               df_new["Billboard Year"] = df_new["Billboard Year"].dt.year
               df_new.loc[df_new["Billboard Year"] > 2050, "Billboard Year"] = df_new["Billboa

               df_new.to_csv("dataset.csv", index=False)
```

12 put_together()

put_together() puts all outliars data into one dataframe.

```
In [2947]: def put_together():
               exit()
           #     df_1 = pd.read_csv("analyze_outliars_1960s.csv")
           #     df_2 = pd.read_csv("analyze_outliars_1970s.csv")
           #     df_3 = pd.read_csv("analyze_outliars_1980s.csv")
           #     df_4 = pd.read_csv("analyze_outliars_1990s.csv")
           #     df_5 = pd.read_csv("analyze_outliars_2000s.csv")
           #     df_6 = pd.read_csv("analyze_outliars_2010s.csv")

           #     df = pd.DataFrame()
           #     df = df.append([df_1, df_2, df_3, df_4, df_5, df_6], sort=False)
           #     df.to_csv("dataset_spotify_with_genre.csv", index=False)
```

13 remove_spotify_zero()

After retriving 'Spotify Popularity' for our dataset.csv list of songs to a dataset_spotify.csv, we removed all songs who has 'Spotify Popularity' = 0, because those songs are not part of Spotify database and we can't impute values for those songs since they may be part of the trend or may be an outliar. Furthermore, this adds years release column by subtracting 2018 by Billboard year.

```
In [2948]: def remove_spotify_zero():
               df = pd.read_csv("dataset_spotify_with_artist_features.csv")
               df = df.loc[df['Spotify Popularity'] != 0]
               df["Years Released"] = 2018 - df["Billboard Year"]
               df["Genre"] = 1
               df.to_csv("dataset_spotify_new.csv", index=False)
```

14 func(x, a, b)

func(x, a, b) represents base function, we used logarithmic model as decay function of years released.

```
In [2949]: def func(x, a, b):
               return a + b*np.log(x)
```

15 train(x_data, y_data)

train(x_data, y_data) performs curve fitting on the x_data and y_date provided.

```
In [2950]: def train(x_data, y_data):
               params, params_covariance = optimize.curve_fit(func, x_data, y_data,
                                                              p0=[1, 1])

               return params, params_covariance
```

16 error(df, params)

error(df, params) calculates error rmse and appends useful error column statistics to the dataframe provided.

```
In [2951]: def error(df, params):
               df['Prediction'] = func(df["Years Released"], params[0], params[1])
               df['Difference'] = abs(df['Prediction'] - df["Spotify Popularity"])
               df['Squared Difference'] = df['Difference']**2
               df['Ratio'] = df['Spotify Popularity']/df['Prediction']

               count = len(df.index)
               rmse = sqrt(df['Squared Difference'].sum()/count)
               print(rmse)
               # print(df['Difference'].describe())
```

```
    df = df.sort_values(['Ratio'])

    return rmse
```

17 best_plot()

best_plot() plots the required plots for this songs project. This function filters the data based on peak rank and observes its decay function as a function of years released (1, 59 where 1 represents the most recent songs). The plot shows individual songs/mean values for each peak rank, and the line of best fit (logarithmic function) produced after training. This function can also calculate error rmse and display overperforming/underperforming threshold.

```
In [2952]: def best_plot():
    df = pd.read_csv("dataset_spotify.csv")
    peak_rank = 1

    df = df.loc[df['Peak Position'] == peak_rank]
    df_grouped = df.groupby(["Years Released"])["Years Released", "Spotify Popularit

    ## Plots individual songs, mean values
    # df.plot.scatter(x='Years Released', y='Spotify Popularity', s=32, rot=90, lab

    fig, ax = plt.subplots()
#     df.plot(kind='scatter', x='value', y='mean', s=60, c='size', cmap='RdYlGn', a
    df.plot.scatter(x='Years Released', y='Spotify Popularity', s=16, label='indivi
                    c=df['Spotify Popularity'], cmap=plt.cm.Spectral, ax=ax)

    # df_grouped.plot.scatter(x='Years Released', y='Spotify Popularity', s=16, rot

    params, params_covariance = train(df_grouped["Years Released"], df_grouped["Spo
    print(params)
    error(df_grouped, params)
    error(df, params)

    x_data = df_grouped["Years Released"]

    plt.plot(x_data, func(x_data, params[0], params[1]), label='best fit')
    plt.plot(x_data, func(x_data, params[0], params[1]) + 20, c="g", label="overper
    plt.plot(x_data, func(x_data, params[0], params[1]) - 20, c="r", label="underpe

    label('Years Released', 'Spotify Popularity Mean', 'Spotify Popularity Mean for
    plt.legend(loc='best')

    plt.savefig("peak" + str(peak_rank) + ".png")
    # df.to_csv("peak" + str(peak_rank) + "outliars.csv", index=False)
```

18 parametrize_rank()

parametrize_rank() trains each peak rank and retrieves appropriate coefficients for each peak rank, which we hope to observe a trend, and parametrize the 100 models for each peak rank into one model.

```
In [2953]: def parametrize_rank():
               df = pd.read_csv("dataset_spotify.csv")
               columns = ['Rank', 'a', 'b']
               df_coeff = pd.DataFrame(columns=columns)

               for i in range(1, 101):
                   df_temp = df.loc[df['Peak Position'] == i]
                   df_grouped = df_temp.groupby(["Years Released"])["Years Released", "Spotify
                   params, params_covariance = train(df_grouped["Years Released"], df_grouped['
                   df_coeff = df_coeff.append({'Rank' : i, 'a' : params[0] , 'b' : params[1]}

               df_coeff.to_csv("coefficients.csv", index=False)
```

19 compare_ranks()

compare_ranks() compares the baseline model for two ranks and see how they differ.

```
In [2954]: def compare_ranks():
               df = pd.read_csv("dataset_spotify.csv")
               df_coeff = pd.read_csv("coefficients.csv")

               rank_one = 64
               rank_two = 65

               df_one = df.loc[df['Peak Position'] == rank_one]
               df_one_grouped = df_one.groupby(["Years Released"])["Years Released", "Spotify
               x_data_one = df_one_grouped["Years Released"]

               df_two = df.loc[df['Peak Position'] == rank_two]
               df_two_grouped = df_two.groupby(["Years Released"])["Years Released", "Spotify
               x_data_two = df_two_grouped["Years Released"]

               plt.plot(x_data_one, func(x_data_one, df_coeff['a'].iloc[rank_one - 1], df_coef
               plt.plot(x_data_two, func(x_data_two, df_coeff['a'].iloc[rank_two - 1], df_coef

               label('Years Released', 'Spotify Popularity Mean', 'Comparing Baseline Models w
               plt.legend(loc='best')

               plt.savefig('comparing_ranks_64_65.png')
```

20 outliars_by_decade()

outliars_by_decade() trains and tests each peak rank and appends 100 dataframes for each peak rank to one dataframe and filters them by its 'Years Released' to get outliars on each decade.

```
In [2955]: def outliars_by_decade():
               df = pd.read_csv("dataset_spotify.csv")
               df_array = []

               for i in range(1, 101):
                   df_temp = df.loc[df['Peak Position'] == i]
                   df_grouped = df_temp.groupby(["Years Released"])["Years Released", "Spotify
                   params, params_covariance = train(df_grouped["Years Released"], df_grouped['
                   error(df_temp, params)
                   df_array.append(df_temp)

               df = pd.DataFrame()
               df = df.append(df_array)

               df_decade = df.loc[df['Years Released'] < 59] # 2000-2009
               df_decade = df_decade.loc[df_decade['Years Released'] > 48] # 2000-2009
               df_decade = df_decade.sort_values(['Ratio'], ascending=False)
               df_decade.to_csv("analyze_outliars_1960s.csv", index=False) # 2000-2009
```

21 label(x_label, y_label, title)

label(x_label, y_label, title) labels the plot appropriate with the right figure sizes.

```
In [2956]: def label(x_label, y_label, title):
               plt.xlabel(x_label, size = 10) # 12
               plt.ylabel(y_label, size = 10)
               plt.xticks(size = 8) # 10
               plt.yticks(size = 8)
               plt.title(title, size = 12) # 15

               rcParams['figure.figsize'] = 7, 4
```

22 func_with_peak(X, a, b, c)

func_with_peak(X, a, b, c) is base model including years released and peak rank.

```
In [2957]: def func_with_peak(X, a, b, c):
               year = X['Years Released']
               peak = X['Peak Position']

               return a + b*np.log(year) + c*np.log(peak)
```

23 error_with_peak(df, params)

error_with_peak(df, params) computes errors for for base model with years release and peak rank.

```
In [2958]: def error_with_peak(df, params):
               df['Prediction'] = func_with_peak(df, params[0], params[1], params[2])
               df['Difference'] = abs(df['Prediction'] - df["Spotify Popularity"])
               df['Squared Difference'] = df['Difference']**2
               df['Ratio'] = df['Spotify Popularity']/df['Prediction']

               count = len(df.index)
               rmse = sqrt(df['Squared Difference'].sum()/count)
               print(rmse)

               df = df.sort_values(['Ratio'])
```

24 train_peak()

train_peak() trains all individual songs on one single base model.

```
In [2959]: def train_peak():
               df = pd.read_csv("dataset_spotify.csv")

               x_data = df[['Years Released','Peak Position']]
               y_data = df["Spotify Popularity"]

               X_train, X_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0

               params, params_covariance = optimize.curve_fit(func_with_peak, X_train, y_train

               df_test = pd.DataFrame(columns = X_train.columns)
               df_test = pd.concat([X_test, y_test], axis=1)

               error_with_peak(df_test, params)
```

25 feature_analysis()

feature_analysis() observes some statistics on artist popularity and genre of the songs in our dataset.

```
In [2960]: def feature_analysis():
               df = pd.read_csv("dataset_spotify_with_artist_genre.csv")

               df_over = df.loc[df['Ratio'] > 2.3]
               df_under = df.loc[df['Ratio'] < 0.1]

               df = df.loc[(df['Ratio'] < 2) & (df['Ratio'] > 0.1)]
```

```
data_one = df['Artist Popularity']
data_two = df_over['Artist Popularity']
data_three = df_under['Artist Popularity']

plt.hist(data_two)
label("Artist Popularity", "Count", "Artist Popularity of Overperforming Songs")

plt.savefig('overperforming_distribution.png')

df = pd.read_csv("analyze_outliars_1970s_modified.csv")
df = df.loc[:25,:]
df = df.groupby('Genre')['Genre'].count()
print(df)

objects = ('Classic', 'Country', 'Folk', 'Disco', 'Funk', 'Jazz', 'Rock')
y_pos = np.arange(len(objects))
performance = [2,1,3,4,1,1,13]

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('Count')
plt.title('Genre of 1970s Overperforming Songs')
rcParams['figure.figsize'] = 9, 5
plt.savefig('genre_overperforming_1970s.png')
```

26 func_with_peak(X, a, b, c, d, e)

func_with_peak(X, a, b, c) is base model including years released, peak rank, artist_popularity, and genre_score.

```
In [2961]: def func_with_all(X, a, b, c, d, e):
               year = X['Years Released']
               peak = X['Peak Position']
               artist_popularity = X['Artist Popularity']
               genre_score = X['Genre Score']

               return a + b*np.log(year) + c*np.log(peak) + d*artist_popularity + e*genre_score
```

27 error_with_peak(df, params)

error_with_peak(df, params) computes errors for for base model with years released, peak rank, artist_popularity, and genre_score.

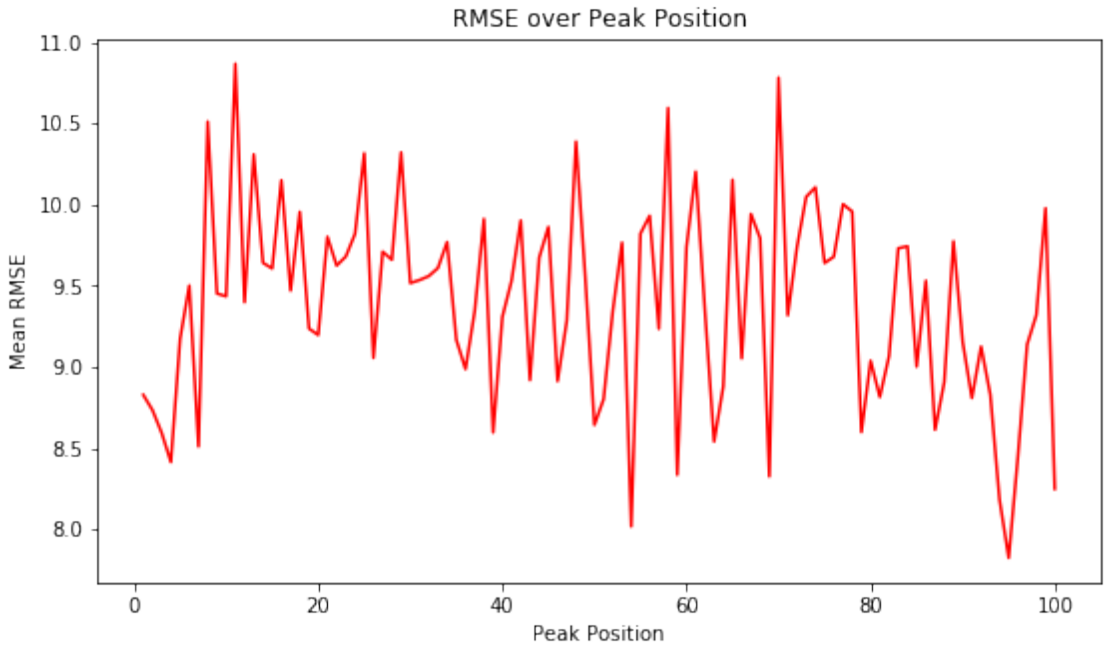
```
In [2962]: def error_with_all(df, params):
               df['Prediction'] = func_with_all(df, params[0], params[1], params[2], params[3]
               df['Difference'] = abs(df['Prediction'] - df["Spotify Popularity"])
```

```
                df['Squared Difference'] = df['Difference']**2
                df['Ratio'] = df['Spotify Popularity']/df['Prediction']

                count = len(df.index)
                rmse = sqrt(df['Squared Difference'].sum()/count)
                print(rmse)

                df = df.sort_values(['Ratio'])
```

28 train_all()

train_all() trains based on features years released, peak rank, artist_popularity, and genre_score.

```
In [2963]: def train_all():
                df = pd.read_csv("dataset_spotify_with_artist_genre.csv")

                x_data = df[['Years Released','Peak Position', 'Artist Popularity', 'Genre Score
                y_data = df["Spotify Popularity"]

                X_train, X_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0

                params, params_covariance = optimize.curve_fit(func_with_all, X_train, y_train,

                df_test = pd.DataFrame(columns = X_train.columns)
                df_test = pd.concat([X_test, y_test], axis=1)

                df_train = pd.DataFrame(columns = X_train.columns)
                df_train = pd.concat([X_train, y_train], axis=1)

                error_with_all(df_train, params)
                error_with_all(df_test, params)

                df_train = df_train.groupby(["Peak Position"])["Peak Position", "Difference"].me
                plt.plot(df_train["Peak Position"], df_train["Difference"], c='r')
                plt.xlabel('Peak Position')
                plt.ylabel('Mean RMSE')
                plt.title('RMSE over Peak Position')
                rcParams['figure.figsize'] = 9, 5
                plt.savefig('rmse_peak.png')

In [2964]: if __name__== "__main__":
                main()

11.743584387902226
11.96461772147108
done
```


RMSE over Peak Position
Mean RMSE
Peak Position
11.0
10.5
10.0
9.5
9.0
8.5
8.0
0
20
40
60
80
100