MATH 484 Project

We are trying to predict the future times and results of sprints and field events at the Olympics.

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import scipy.stats as stats
import numpy as np
import statsmodels.api as sm
import statsmodels.nonparametric as snp
import statsmodels.stats as sms
import scipy.linalg as lg
import plotly

track_results = pd.read_csv('results.csv')
```

In [2]:
```
# Creating separate dictionaries for each event.
tf_results = dict(tuple(track_results.groupby("Event")))
```

I am going to create dataframes and plots of the sprint events from the Summer Olympics that we have results from.

In [3]:
```
print(track_results.Event.unique())
```

```
['10000M Men' '100M Men' '110M Hurdles Men' '1500M Men' '200M Men'
 '20Km Race Walk Men' '3000M Steeplechase Men' '400M Hurdles Men'
 '400M Men' '4X100M Relay Men' '4X400M Relay Men' '5000M Men'
 '50Km Race Walk Men' '800M Men' 'Decathlon Men' 'Discus Throw Men'
 'Hammer Throw Men' 'High Jump Men' 'Javelin Throw Men' 'Long Jump Men'
 'Marathon Men' 'Pole Vault Men' 'Shot Put Men' 'Triple Jump Men'
 '10000M Women' '100M Hurdles Women' '100M Women' '1500M Women'
 '200M Women' '20Km Race Walk Women' '3000M Steeplechase Women'
 '400M Hurdles Women' '400M Women' '4X100M Relay Women'
 '4X400M Relay Women' '5000M Women' '800M Women' 'Discus Throw Women'
 'Hammer Throw Women' 'Heptathlon Women' 'High Jump Women'
 'Javelin Throw Women' 'Long Jump Women' 'Marathon Women'
 'Pole Vault Women' 'Shot Put Women' 'Triple Jump Women']
```

We will create results scatter plots for field events as well overtime at the Olympics.

Simple Linear Regression Models for 100 Meter, 200 Meter, Long Jump, Shot Put, 1500 M Run

We start by running a simple linear regression for the 100 M Men and Women's Events.

In [4]:
```
Men_100M = tf_results['100M Men']
Men_100M.head()
```

Out[4]:

	Gender	Event	Location	Year	Medal	Name	Nationality	Result
69	M	100M Men	Rio	2016	G	Usain BOLT	JAM	9.81
70	M	100M Men	Rio	2016	S	Justin GATLIN	USA	9.89
71	M	100M Men	Rio	2016	B	Andre DE GRASSE	CAN	9.91
72	M	100M Men	Beijing	2008	G	Usain BOLT	JAM	9.69
73	M	100M Men	Beijing	2008	S	Richard THOMPSON	TTO	9.89

In [5]:
```
X = tf_results['100M Men'].Year.astype(float)
y = tf_results['100M Men'].Result.astype(float)
X = sm.add_constant(X)
model_100M_Men = sm.OLS(y, X).fit()
print(model_100M_Men.params)
model_100M_Men.summary()
```

```
const     37.670494
Year      -0.013918
dtype: float64
```

Out[5]:

OLS Regression Results

Dep. Variable:	Result	**R-squared:**	0.736
Model:	OLS	**Adj. R-squared:**	0.732
Method:	Least Squares	**F-statistic:**	217.1
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	3.03e-24
Time:	15:11:13	**Log-Likelihood:**	-19.177
No. Observations:	80	**AIC:**	42.35
Df Residuals:	78	**BIC:**	47.12
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	37.6705	1.850	20.366	0.000	33.988	41.353
Year	-0.0139	0.001	-14.735	0.000	-0.016	-0.012

Omnibus:	56.123	**Durbin-Watson:**	0.350
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	239.787
Skew:	2.221	**Prob(JB):**	8.53e-53
Kurtosis:	10.226	**Cond. No.**	1.04e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.04e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [6]:
```
X = tf_results['100M Women'].Year.astype(float)
y = tf_results['100M Women'].Result.astype(float)
X = sm.add_constant(X)
model_100M_Women = sm.OLS(y, X).fit()
print(model_100M_Women.params)
model_100M_Women.summary()
```

```
const    39.259190
Year     -0.014167
dtype: float64
```

Out[6]:

OLS Regression Results

Dep. Variable:	Result	**R-squared:**	0.763
Model:	OLS	**Adj. R-squared:**	0.758
Method:	Least Squares	**F-statistic:**	179.9
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	3.93e-19
Time:	15:11:13	**Log-Likelihood:**	10.628
No. Observations:	58	**AIC:**	-17.26
Df Residuals:	56	**BIC:**	-13.14
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	39.2592	2.087	18.811	0.000	35.078	43.440
Year	-0.0142	0.001	-13.413	0.000	-0.016	-0.012

Omnibus:	1.636	**Durbin-Watson:**	0.818
Prob(Omnibus):	0.441	**Jarque-Bera (JB):**	0.926
Skew:	0.254	**Prob(JB):**	0.629
Kurtosis:	3.353	**Cond. No.**	1.53e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.53e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

200 M Men's and Women's Events now:

In [8]:
```
X = tf_results['200M Men'].Year.astype(float)
y = tf_results['200M Men'].Result.astype(float)
X = sm.add_constant(X)
model_200M_Men = sm.OLS(y, X).fit()
print(model_200M_Men.params)
model_200M_Men.summary()
```

```
const    67.599404
Year     -0.023895
dtype: float64
```

Out[8]:

OLS Regression Results

Dep. Variable:	Result	**R-squared:**	0.866
Model:	OLS	**Adj. R-squared:**	0.864
Method:	Least Squares	**F-statistic:**	471.6
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	1.40e-33
Time:	15:11:13	**Log-Likelihood:**	-21.997
No. Observations:	75	**AIC:**	47.99
Df Residuals:	73	**BIC:**	52.63
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	67.5994	2.159	31.310	0.000	63.297	71.902
Year	-0.0239	0.001	-21.716	0.000	-0.026	-0.022

Omnibus:	0.287	**Durbin-Watson:**	1.042
Prob(Omnibus):	0.866	**Jarque-Bera (JB):**	0.470
Skew:	0.059	**Prob(JB):**	0.791
Kurtosis:	2.631	**Cond. No.**	1.12e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.12e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [9]:
```
X = tf_results['200M Women'].Year.astype(float)
y = tf_results['200M Women'].Result.astype(float)
X = sm.add_constant(X)
model_200M_Women = sm.OLS(y, X).fit()
print(model_200M_Women.params)
model_200M_Women.summary()
```

```
const    94.100684
Year     -0.036020
dtype: float64
```

Out[9]:

OLS Regression Results

Dep. Variable:	Result	**R-squared:**	0.684
Model:	OLS	**Adj. R-squared:**	0.678
Method:	Least Squares	**F-statistic:**	106.2
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	7.39e-14
Time:	15:11:13	**Log-Likelihood:**	-39.140
No. Observations:	51	**AIC:**	82.28
Df Residuals:	49	**BIC:**	86.14
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	94.1007	6.928	13.583	0.000	80.178	108.023
Year	-0.0360	0.003	-10.303	0.000	-0.043	-0.029

Omnibus:	2.396	**Durbin-Watson:**	0.507
Prob(Omnibus):	0.302	**Jarque-Bera (JB):**	2.287
Skew:	0.489	**Prob(JB):**	0.319
Kurtosis:	2.651	**Cond. No.**	1.84e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.84e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [11]:
```
X = tf_results['Long Jump Men'].Year.astype(float)
y = tf_results['Long Jump Men'].Result.astype(float)
X = sm.add_constant(X)
LJ_Men = sm.OLS(y, X).fit()
print(LJ_Men.params)
LJ_Men.summary()
```

```
const   -18.587442
Year      0.013485
dtype: float64
```

Out[11]:

OLS Regression Results

Dep. Variable:	Result	**R-squared:**	0.792
Model:	OLS	**Adj. R-squared:**	0.788
Method:	Least Squares	**F-statistic:**	231.9
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	1.87e-22
Time:	15:11:14	**Log-Likelihood:**	-8.8863
No. Observations:	63	**AIC:**	21.77
Df Residuals:	61	**BIC:**	26.06
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-18.5874	1.733	-10.723	0.000	-22.054	-15.121
Year	0.0135	0.001	15.229	0.000	0.012	0.015

Omnibus:	10.035	**Durbin-Watson:**	0.888
Prob(Omnibus):	0.007	**Jarque-Bera (JB):**	9.869
Skew:	-0.823	**Prob(JB):**	0.00719
Kurtosis:	4.024	**Cond. No.**	9.51e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.51e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [12]:
```
X = tf_results['Long Jump Women'].Year.astype(float)
y = tf_results['Long Jump Women'].Result.astype(float)
X = sm.add_constant(X)
LJ_Women = sm.OLS(y, X).fit()
print(LJ_Women.params)
LJ_Women.summary()
```

```
const   -21.278026
Year      0.014137
dtype: float64
```

Out[12]:

OLS Regression Results

Dep. Variable:	Result	**R-squared:**	0.734
Model:	OLS	**Adj. R-squared:**	0.726
Method:	Least Squares	**F-statistic:**	90.97
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	5.22e-11
Time:	15:11:14	**Log-Likelihood:**	11.192
No. Observations:	35	**AIC:**	-18.38
Df Residuals:	33	**BIC:**	-15.27
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-21.2780	2.947	-7.221	0.000	-27.273	-15.283
Year	0.0141	0.001	9.538	0.000	0.011	0.017

Omnibus:	0.431	**Durbin-Watson:**	0.745
Prob(Omnibus):	0.806	**Jarque-Bera (JB):**	0.537
Skew:	0.228	**Prob(JB):**	0.765
Kurtosis:	2.601	**Cond. No.**	1.91e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.91e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [14]:
```
X = tf_results['Shot Put Men'].Year.astype(float)
y = tf_results['Shot Put Men'].Result.astype(float)
X = sm.add_constant(X)
SP_Men = sm.OLS(y, X).fit()
print(SP_Men.params)
SP_Men.summary()
```

```
const   -142.645216
Year       0.082039
dtype: float64
```

Out[14]:

OLS Regression Results

Dep. Variable:	Result	**R-squared:**	0.930
Model:	OLS	**Adj. R-squared:**	0.929
Method:	Least Squares	**F-statistic:**	774.3
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	3.06e-35
Time:	15:11:14	**Log-Likelihood:**	-80.094
No. Observations:	60	**AIC:**	164.2
Df Residuals:	58	**BIC:**	168.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-142.6452	5.777	-24.692	0.000	-154.209	-131.081
Year	0.0820	0.003	27.826	0.000	0.076	0.088

Omnibus:	0.027	**Durbin-Watson:**	0.658
Prob(Omnibus):	0.986	**Jarque-Bera (JB):**	0.152
Skew:	-0.045	**Prob(JB):**	0.927
Kurtosis:	2.770	**Cond. No.**	9.38e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.38e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [15]:
```
X = tf_results['Shot Put Women'].Year.astype(float)
y = tf_results['Shot Put Women'].Result.astype(float)
X = sm.add_constant(X)
SP_Women = sm.OLS(y, X).fit()
print(SP_Women.params)
SP_Women.summary()
```

```
const   -103.642325
Year       0.062040
dtype: float64
```

Out[15]:

OLS Regression Results

Dep. Variable:	Result	**R-squared:**	0.411
Model:	OLS	**Adj. R-squared:**	0.396
Method:	Least Squares	**F-statistic:**	27.24
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	6.25e-06
Time:	15:11:14	**Log-Likelihood:**	-72.319
No. Observations:	41	**AIC:**	148.6
Df Residuals:	39	**BIC:**	152.1
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-103.6423	23.576	-4.396	0.000	-151.329	-55.955
Year	0.0620	0.012	5.220	0.000	0.038	0.086

Omnibus:	0.304	**Durbin-Watson:**	0.593
Prob(Omnibus):	0.859	**Jarque-Bera (JB):**	0.445
Skew:	0.173	**Prob(JB):**	0.800
Kurtosis:	2.625	**Cond. No.**	2.07e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.07e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [17]:
```
#Men's 1500M
men_1500m = pd.read_csv('Men 1500M.csv')
X = men_1500m['Year'].astype(float)
y = men_1500m['seconds'].astype(float)
X = sm.add_constant(X)
Men_1500M = sm.OLS(y, X).fit()
print(Men_1500M.params)
Men_1500M.summary()
```

```
const     846.904980
Year       -0.316465
dtype: float64
```

Out[17]:

OLS Regression Results

Dep. Variable:	seconds	**R-squared:**	0.682
Model:	OLS	**Adj. R-squared:**	0.677
Method:	Least Squares	**F-statistic:**	164.8
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	8.04e-21
Time:	15:11:14	**Log-Likelihood:**	-275.71
No. Observations:	79	**AIC:**	555.4
Df Residuals:	77	**BIC:**	560.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	846.9050	48.285	17.540	0.000	750.756	943.053
Year	-0.3165	0.025	-12.837	0.000	-0.366	-0.267

Omnibus:	48.286	**Durbin-Watson:**	0.292
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	132.075
Skew:	2.138	**Prob(JB):**	2.09e-29
Kurtosis:	7.673	**Cond. No.**	1.05e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.05e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [18]:
```
#Women's 1500M
women_1500m = pd.read_csv('Women 1500M.csv')
women_1500m.head(3)
X = women_1500m['Year'].astype(float)
y = women_1500m['seconds'].astype(float)
X = sm.add_constant(X)
Women_1500M = sm.OLS(y, X).fit()
print(Women_1500M.params)
Women_1500M.summary()
```

```
const    40.274876
Year      0.101555
dtype: float64
```

Out[18]:

OLS Regression Results

Dep. Variable:	seconds	**R-squared:**	0.110
Model:	OLS	**Adj. R-squared:**	0.081
Method:	Least Squares	**F-statistic:**	3.722
Date:	Thu, 29 Nov 2018	**Prob (F-statistic):**	0.0632
Time:	15:11:14	**Log-Likelihood:**	-90.462
No. Observations:	32	**AIC:**	184.9
Df Residuals:	30	**BIC:**	187.9
Df Model:	1		
Covariance Type:	nonrobust		

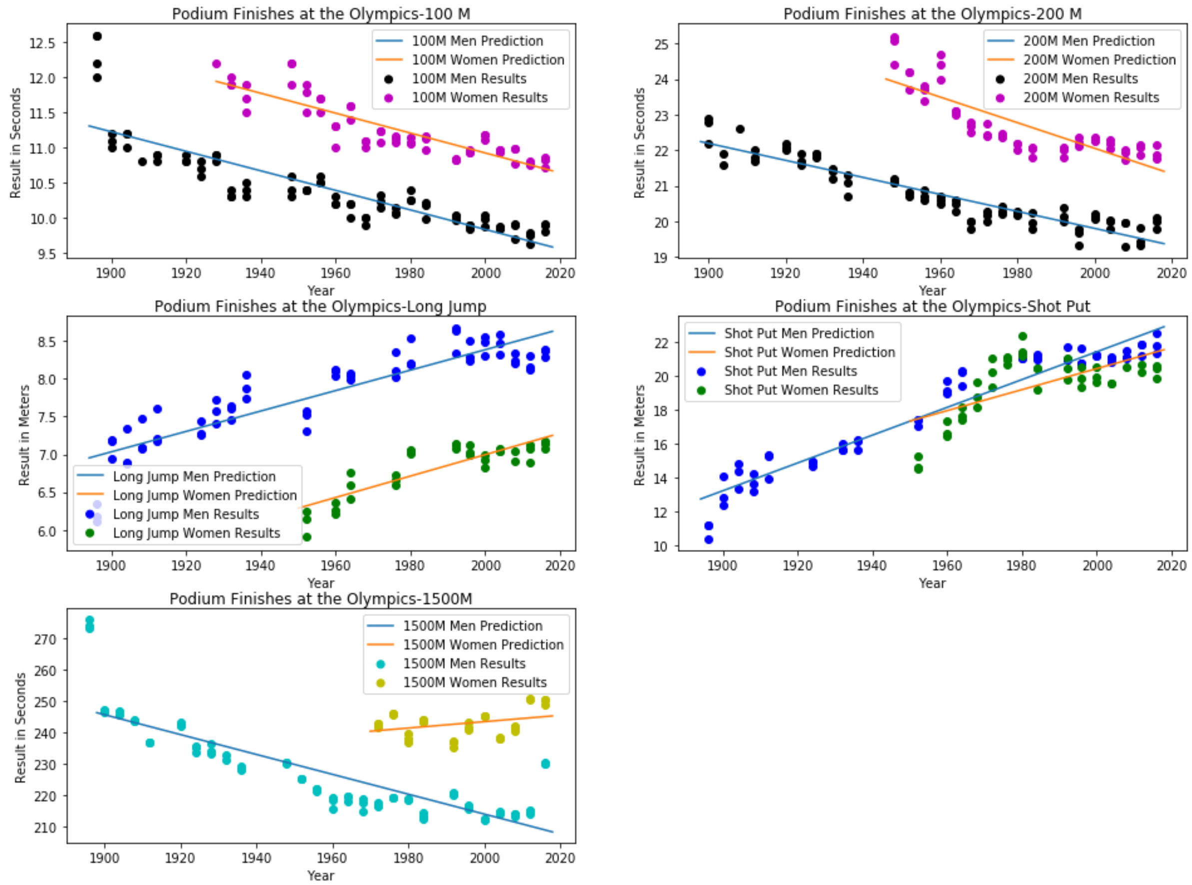
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	40.2749	104.971	0.384	0.704	-174.104	254.653
Year	0.1016	0.053	1.929	0.063	-0.006	0.209

Omnibus:	8.608	**Durbin-Watson:**	0.832
Prob(Omnibus):	0.014	**Jarque-Bera (JB):**	2.354
Skew:	-0.147	**Prob(JB):**	0.308
Kurtosis:	1.704	**Cond. No.**	2.80e+05

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.8e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [132]:
```
#Regression Subplots
plt.figure(figsize=(16, 12))
plt.subplot(321)
time_m = np.linspace(1894, 2018, num = 2000)
time_w = np.linspace(1928, 2018, num = 2000)
Men_pred = 37.670494 - 0.013918 * time_m
Women_pred =  39.259190 - 0.014167 * time_w
plt.scatter(tf_results['100M Men'].Year, tf_results['100M Men'].Result, color
= 'k')
plt.plot(time_m, Men_pred)
plt.scatter(tf_results['100M Women'].Year, tf_results['100M Women'].Result, co
lor = 'm')
plt.plot(time_w, Women_pred)
plt.xlabel('Year')
plt.ylabel('Result in Seconds')
plt.title('Podium Finishes at the Olympics-100 M')
plt.legend(['100M Men Prediction', '100M Women Prediction', '100M Men Results'
, '100M Women Results'])


plt.subplot(322)
time_m = np.linspace(1898, 2018, num = 2000)
time_w = np.linspace(1946, 2018, num = 2000)
Men_pred = 67.599404 -0.023895 * time_m
Women_pred =  94.100684 -0.036020  * time_w
plt.scatter(tf_results['200M Men'].Year, tf_results['200M Men'].Result, color
= 'k')
plt.plot(time_m, Men_pred)
plt.scatter(tf_results['200M Women'].Year, tf_results['200M Women'].Result, co
lor = 'm')
plt.plot(time_w, Women_pred)
plt.xlabel('Year')
plt.ylabel('Result in Seconds')
plt.title('Podium Finishes at the Olympics-200 M')
plt.legend(['200M Men Prediction', '200M Women Prediction', '200M Men Results'
, '200M Women Results'])

plt.subplot(323)
time_m = np.linspace(1894, 2018, num = 2000)
time_w = np.linspace(1950, 2018, num = 2000)
Men_pred = -18.587442 +0.013485 * time_m
Women_pred = -21.278026 +0.014137  * time_w
plt.scatter(tf_results['Long Jump Men'].Year, tf_results['Long Jump Men'].Resu
lt, color = 'b')
plt.plot(time_m, Men_pred)
plt.scatter(tf_results['Long Jump Women'].Year, tf_results['Long Jump Women'].
Result, color = 'g')
plt.plot(time_w, Women_pred)
plt.xlabel('Year')
plt.ylabel('Result in Meters')
plt.title('Podium Finishes at the Olympics-Long Jump')
plt.legend(['Long Jump Men Prediction', 'Long Jump Women Prediction', 'Long Ju
mp Men Results', 'Long Jump Women Results'])


plt.subplot(324)
```

```
time_m = np.linspace(1894, 2018, num = 2000)
time_w = np.linspace(1950, 2018, num = 2000)
Men_pred = -142.645216 +0.082039 * time_m
Women_pred = -103.642325 +0.062040  * time_w
plt.scatter(tf_results['Shot Put Men'].Year, tf_results['Shot Put Men'].Result
, color = 'b')
plt.plot(time_m, Men_pred)
plt.scatter(tf_results['Shot Put Women'].Year, tf_results['Shot Put Women'].Re
sult, color = 'g')
plt.plot(time_w, Women_pred)
plt.xlabel('Year')
plt.ylabel('Result in Meters')
plt.title('Podium Finishes at the Olympics-Shot Put')
plt.legend(['Shot Put Men Prediction', 'Shot Put Women Prediction', 'Shot Put
 Men Results', 'Shot Put Women Results'])

plt.subplot(325)
time_m = np.linspace(1898, 2018, num = 2000)
time_w = np.linspace(1970, 2018, num = 2000)
Men_pred = 846.904980 -0.316465 * time_m
Women_pred = 40.274876 +0.101555  * time_w
plt.scatter(men_1500m['Year'], men_1500m['seconds'], color = 'c')
plt.plot(time_m, Men_pred)
plt.scatter(women_1500m['Year'], women_1500m['seconds'], color = 'y')
plt.plot(time_w, Women_pred)
plt.xlabel('Year')
plt.ylabel('Result in Seconds')
plt.title('Podium Finishes at the Olympics-1500M')
plt.legend(['1500M Men Prediction', '1500M Women Prediction', '1500M Men Resul
ts', '1500M Women Results'])


plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None,
 hspace=0.25)
plt.savefig('slr_results.png')
plt.show()
```

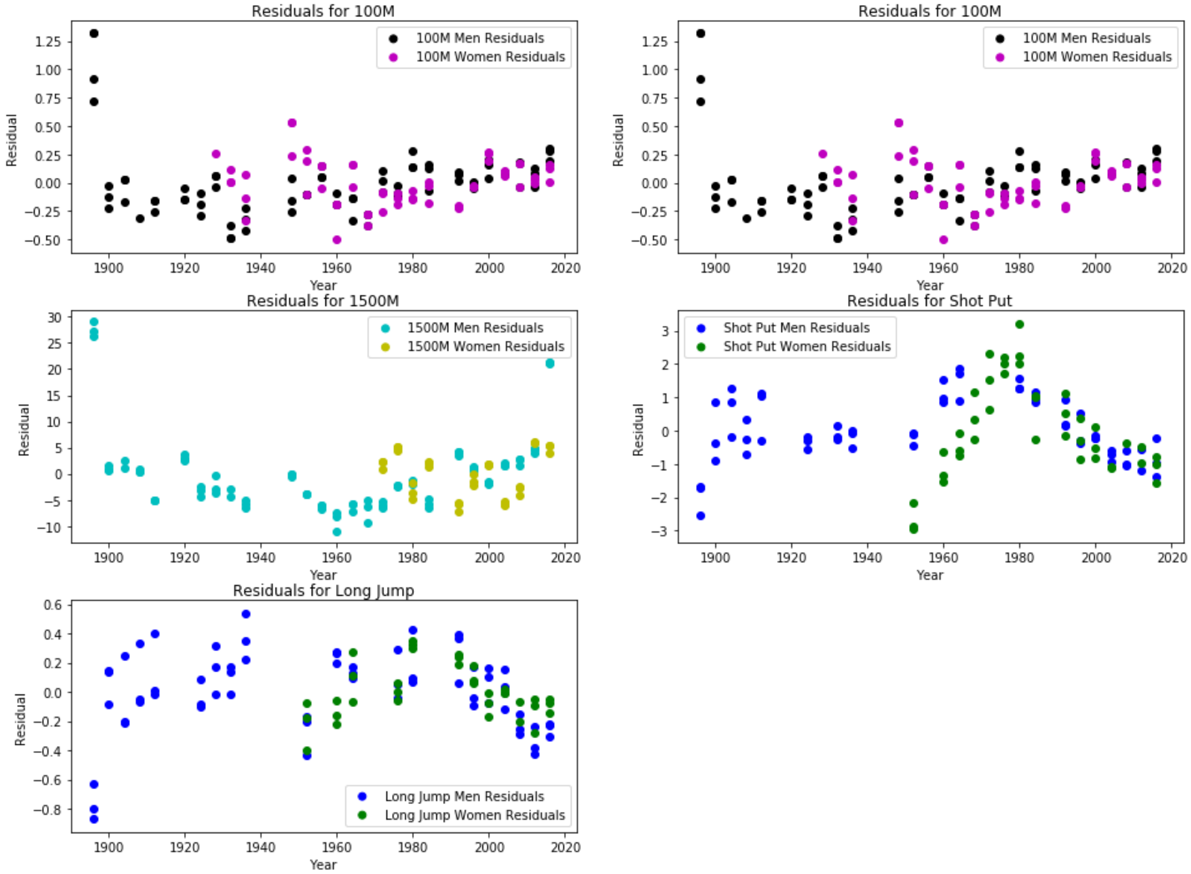


Diagonostics of the Simple Linear Models

We now will look at the residuals to try to determine if we can make the assumption that the error is normally distributed with mean 0 and variance σ^2, and run some diagnostic tests to determine whether the error is constant.

In [91]:
```
#Residuals for 100M
res_100_m = tf_results['100M Men'].Result.astype(float) - \
(37.670494 - 0.013918 * tf_results['100M Men'].Year.astype(float))
res_100_w =  tf_results['100M Women'].Result.astype(float) - \
(39.259190 - 0.014167 * tf_results['100M Women'].Year.astype(float))

#Residuals for 200M
res_200_m = tf_results['200M Men'].Result.astype(float) - \
(67.599404 -0.023895 * tf_results['200M Men'].Year.astype(float))
res_200_w = tf_results['200M Women'].Result.astype(float) - \
(94.100684 -0.036020  * tf_results['200M Women'].Year.astype(float))

#Residuals for Long Jump
res_lj_m = tf_results['Long Jump Men'].Result.astype(float) - \
(-18.587442 +0.013485 * tf_results['Long Jump Men'].Year.astype(float))
res_lj_w = tf_results['Long Jump Women'].Result.astype(float) - \
(-21.278026 +0.014137  * tf_results['Long Jump Women'].Year.astype(float))

#Residuals for Shot Put
res_sp_m = tf_results['Shot Put Men'].Result.astype(float) - \
(-142.645216 +0.082039 * tf_results['Shot Put Men'].Year.astype(float))
res_sp_w = tf_results['Shot Put Women'].Result.astype(float) - \
(-103.642325 +0.062040  * tf_results['Shot Put Women'].Year.astype(float))

#Residuals for 1500M
res_1500_m = men_1500m['seconds'] - (846.904980 -0.316465 * men_1500m['Year'])
res_1500_w = women_1500m['seconds'] - (40.274876 +0.101555 * women_1500m['Yea
r'])
```

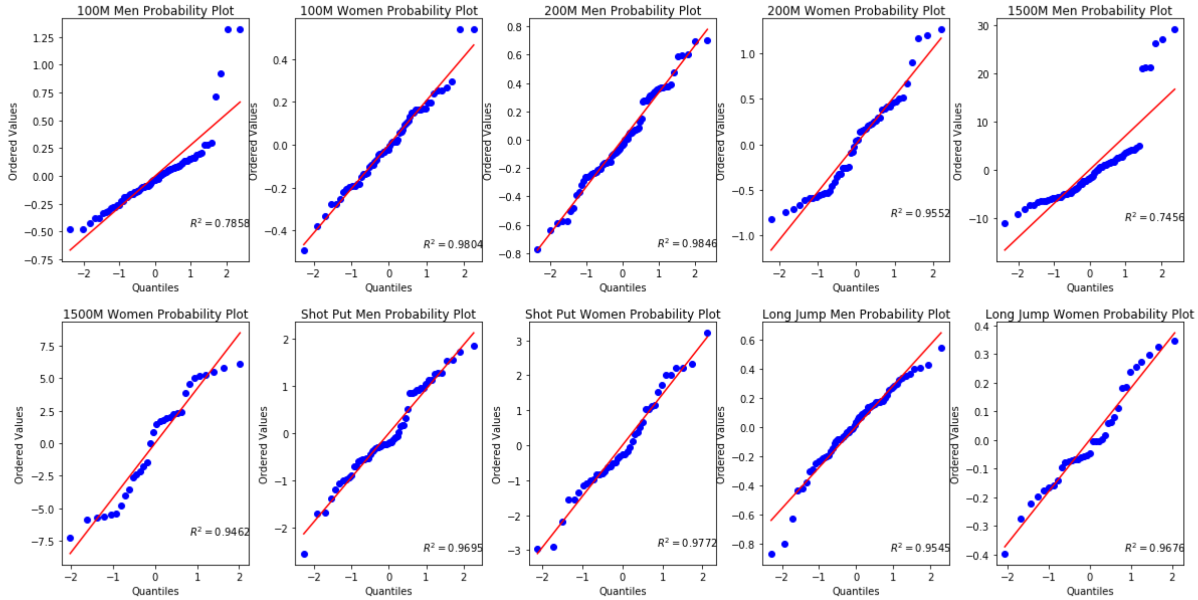
In [131]:
```
#Residual Plots for Simple Linear Models
plt.figure(figsize = (16, 12))
plt.subplot(321)
plt.scatter(tf_results['100M Men'].Year, res_100_m, color = 'k')
plt.scatter(tf_results['100M Women'].Year, res_100_w, color = 'm')
plt.xlabel('Year')
plt.ylabel('Residual')
plt.title('Residuals for 100M')
plt.legend(['100M Men Residuals', '100M Women Residuals'])

plt.subplot(322)
plt.scatter(tf_results['100M Men'].Year, res_100_m, color = 'k')
plt.scatter(tf_results['100M Women'].Year, res_100_w, color = 'm')
plt.xlabel('Year')
plt.ylabel('Residual')
plt.title('Residuals for 100M')
plt.legend(['100M Men Residuals', '100M Women Residuals'])

plt.subplot(323)
plt.scatter(men_1500m['Year'], res_1500_m, color = 'c')
plt.scatter(women_1500m['Year'], res_1500_w, color = 'y')
plt.xlabel('Year')
plt.ylabel('Residual')
plt.title('Residuals for 1500M')
plt.legend(['1500M Men Residuals', '1500M Women Residuals'])

plt.subplot(324)
plt.scatter(tf_results['Shot Put Men'].Year, res_sp_m, color = 'b')
plt.scatter(tf_results['Shot Put Women'].Year, res_sp_w, color = 'g')
plt.xlabel('Year')
plt.ylabel('Residual')
plt.title('Residuals for Shot Put')
plt.legend(['Shot Put Men Residuals', 'Shot Put Women Residuals'])

plt.subplot(325)
plt.scatter(tf_results['Long Jump Men'].Year, res_lj_m, color = 'b')
plt.scatter(tf_results['Long Jump Women'].Year, res_lj_w, color = 'g')
plt.xlabel('Year')
plt.ylabel('Residual')
plt.title('Residuals for Long Jump')
plt.legend(['Long Jump Men Residuals', 'Long Jump Women Residuals'])

plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None,
 hspace=0.25)
plt.savefig('slr_residual_results.png')
plt.show()
```



We are somewhat suspicious of whether or not the error variance is constant or normally distributed. We also believe that we may need higher order terms or a Box-Cox transformation to fit our regression model better than it currently fits the data.

Normal Probability Plots

In [130]:
```
#Normal Probability Plots
plt.figure(figsize=(20, 10))
plt.subplot(251)
#100M Men
res_prob_100m_m = stats.probplot(res_100_m, plot= plt)
plt.title('100M Men Probability Plot')

plt.subplot(252)
#100M Women
res_prob_100m_w = stats.probplot(res_100_w, plot= plt)
plt.title('100M Women Probability Plot')

plt.subplot(253)
#200M Men
res_prob_200m_m = stats.probplot(res_200_m, plot= plt)
plt.title('200M Men Probability Plot')

plt.subplot(254)
#200M Women
res_prob_200m_w = stats.probplot(res_200_w, plot= plt)
plt.title('200M Women Probability Plot')

plt.subplot(255)
#1500M Men
res_prob_1500m_m = stats.probplot(res_1500_m, plot= plt)
plt.title('1500M Men Probability Plot')

plt.subplot(256)
#1500M Women
res_prob_1500m_w = stats.probplot(res_1500_w, plot= plt)
plt.title('1500M Women Probability Plot')

plt.subplot(257)
#Men's Shot Put
res_prob_sp_m = stats.probplot(res_sp_m, plot= plt)
plt.title('Shot Put Men Probability Plot')

plt.subplot(258)
#Women's Shot Put
res_prob_sp_w = stats.probplot(res_sp_w, plot= plt)
plt.title('Shot Put Women Probability Plot')

plt.subplot(259)
#Men's Long Jump
res_prob_lj_m = stats.probplot(res_lj_m, plot= plt)
plt.title('Long Jump Men Probability Plot')

plt.subplot(2,5,10)
#Women's Long Jump
res_prob_lj_w = stats.probplot(res_lj_w, plot= plt)
plt.title('Long Jump Women Probability Plot')
plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=0.25,
 hspace=0.25)
plt.savefig('slr_normprob_results.png')
plt.show()
```



It appears that there are some departures in normality in every single normal probability plot in varying degrees of seriousness except for Shot Put, Women's Long Jump, and Men's 200M dash. This again strongly implies that the error variance is not constant for this model and that we may need to utilize some higher order terms to better fit the data.

Bruesch-Pagan Tests for Constant Error Variance

We will also run a Bruesch-Pagan test to test form departures in constancy of the error terms. Remember, this test is carried out as follows:

We regress the function: $log(\sigma_i^2) = \gamma_0 + \gamma_1 x_{i1}$

$H_0 : \gamma_1 = 0$

$H_1 : \gamma_1 \neq 0$

Test statistic: $\frac{SSR^*/2}{(SSE/n)^2}$

Where SSR^* is the regression sum of squares for the log regression of the residuals.

Critical value for the rejection region would be $\chi^2_{\alpha/2;1}$, and we will test at an α = 0.05 level.

In [107]:
```
from statsmodels.stats import diagnostic as dn
bp_100_m = dn.het_breuschpagan(model_100M_Men.resid, model_100M_Men.model.exog
)
print("The Breusch-Pagan test yields a p-value of: ", bp_100_m[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.00066013566230912351, '.')
```

In [108]:
```
bp_100_w = dn.het_breuschpagan(model_100M_Women.resid, model_100M_Women.model.
exog)
print("The Breusch-Pagan test yields a p-value of: ", bp_100_w[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.021171634781806045, '.')
```

In [109]:
```
bp_200_m = dn.het_breuschpagan(model_200M_Men.resid, model_200M_Men.model.exog
)
print("The Breusch-Pagan test yields a p-value of: ", bp_200_m[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.82010810471300388, '.')
```

In [110]:
```
bp_200_w = dn.het_breuschpagan(model_200M_Women.resid, model_200M_Women.model.
exog)
print("The Breusch-Pagan test yields a p-value of: ", bp_200_w[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.0047673629262334448, '.')
```

In [111]:
```
bp_1500_m = dn.het_breuschpagan(Men_1500M.resid, Men_1500M.model.exog)
print("The Breusch-Pagan test yields a p-value of: ", bp_1500_m[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.25447848550659419, '.')
```

In [112]:
```
bp_1500_w = dn.het_breuschpagan(Women_1500M.resid, Women_1500M.model.exog)
print("The Breusch-Pagan test yields a p-value of: ", bp_1500_w[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.10160731583559358, '.')
```

In [113]:
```
bp_lj_m = dn.het_breuschpagan(LJ_Men.resid, LJ_Men.model.exog)
print("The Breusch-Pagan test yields a p-value of: ", bp_lj_m[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.04485189184383321, '.')
```

In [114]:
```
bp_lj_w = dn.het_breuschpagan(LJ_Women.resid, LJ_Women.model.exog)
print("The Breusch-Pagan test yields a p-value of: ", bp_lj_w[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.073930238062038287, '.')
```

In [115]:
```
bp_sp_m = dn.het_breuschpagan(SP_Men.resid, SP_Men.model.exog)
print("The Breusch-Pagan test yields a p-value of: ", bp_sp_m[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.25290089457670523, '.')
```

In [116]:
```
bp_sp_w = dn.het_breuschpagan(SP_Women.resid, SP_Women.model.exog)
print("The Breusch-Pagan test yields a p-value of: ", bp_sp_w[3],".")
```

```
('The Breusch-Pagan test yields a p-value of: ', 0.0072550835390873597, '.')
```

Based on the Bruesch-Pagan tests we have run, it appears that the only track results that we have regressed that do not have constant error variance are the 100M Results, the Women's 200M, the Men's Long Jump, and the Women's Shot Put at an significance level of $\alpha = 0.05$.

Goodness of Fit Tests on First Order Model:

Since we have repeat observations at each independent predictor variable, we should be able to run an F test for lack of fit. Remember, the F-test for lack of fit is carried out as follows:

$H_0 : E(Y) = \beta_0 + \beta_1 X$

$H_1 : E(Y) \neq \beta_0 + \beta_1 X$

The test statistic here would be $F^* = \frac{SSLF}{c-2} / \frac{SSPE}{n-c} = \frac{MSLF}{MSPE}$, where c is the number of years with replicates and $n = \sum j = 1^c n_j$ and n is the number of observations.

Our decision rule is:

$F^* \leq F(1-\alpha; c-2, n-c)$ then we conclude that our simple linear model is appropriate.

$F^* > F(1-\alpha; c-2, n-c)$ then we conclude that our simple linear model is not appropriate for our data.

We will control the error at α = 0.05.

ALL THE CODE WILL BE IN R FOR THESE TESTS.

All of the results are on Github. In all cases we conclude that the simple linear models are not appropriate. Clearly, we should use some Box-Cox transformations on our results in order to better predict the results.