In [21]:
```
# import data analysis library
import pandas as pd

# import numerical computing library (arrays, matrices, etc.)
import numpy as np

# import plotting library
from matplotlib.pyplot import subplots

# import operating system library
import os

# import statistical computing library
import statsmodels.api as sm

# import VIF and ANOVA functions
from statsmodels.stats.outliers_influence import variance_inflation_factor a
from statsmodels.stats.anova import anova_lm

# import data loading
from ISLP import load_data

# import model methods
from ISLP.models import (ModelSpec as MS, summarize, poly)
```

In [22]:
```
# specify desired path
path = "/Users/karanvirmann/Desktop"

# change working directory
os.chdir(path)

# check if current working directory changed
retval = os.getcwd()
print("%s" % retval)
```

```
/Users/karanvirmann/Desktop
```

In [23]:
```
# load training data
combine = pd.read_csv('training_combine_data.csv')

# display training data
combine
```

Out[23]:

	Player	Pos	School	College	Ht	Wt	40yd	Vertical	Bench	Broa Jum
0	Israel Abanikanda	RB	Pittsburgh	College Stats	10-May	216	NaN	NaN	NaN	Na
1	Yasir Abdullah	LB	Louisville	College Stats	01-Jun	237	4.47	36.5	NaN	129
2	Devon Achane	RB	Texas A&M	College Stats	09-May	188	4.32	33.0	NaN	Na
3	Jordan Addison	WR	USC	College Stats	11-May	173	4.49	34.0	NaN	122
4	Adetomiwa Adebawore	DE	Northwestern	College Stats	02-Jun	282	4.49	37.5	27.0	125
...	...	...	...	...	...	...	...	...	...	
314	Luke Wypler	C	Ohio St.	College Stats	03-Jun	303	5.14	30.5	NaN	106
315	Bryce Young	QB	Alabama	College Stats	10-May	204	NaN	NaN	NaN	Na
316	Byron Young	DT	Alabama	College Stats	03-Jun	294	NaN	26.0	24.0	108
317	Byron Young	EDGE	Tennessee	College Stats	02-Jun	250	4.43	38.0	22.0	132
318	Cameron Young	DT	Mississippi St.	College Stats	03-Jun	304	5.10	NaN	NaN	Na

319 rows × 14 columns

In [24]:
```
# restrict training data to include only columns for player, 40 yd, vertical
combine = combine[['Player', 'Wt', '40yd', 'Vertical', 'Broad Jump']]

# removing observations with no measurement for 40yd, vertical, broad jump a
combine = combine.dropna()

# check if data presentation is as desired
combine
```

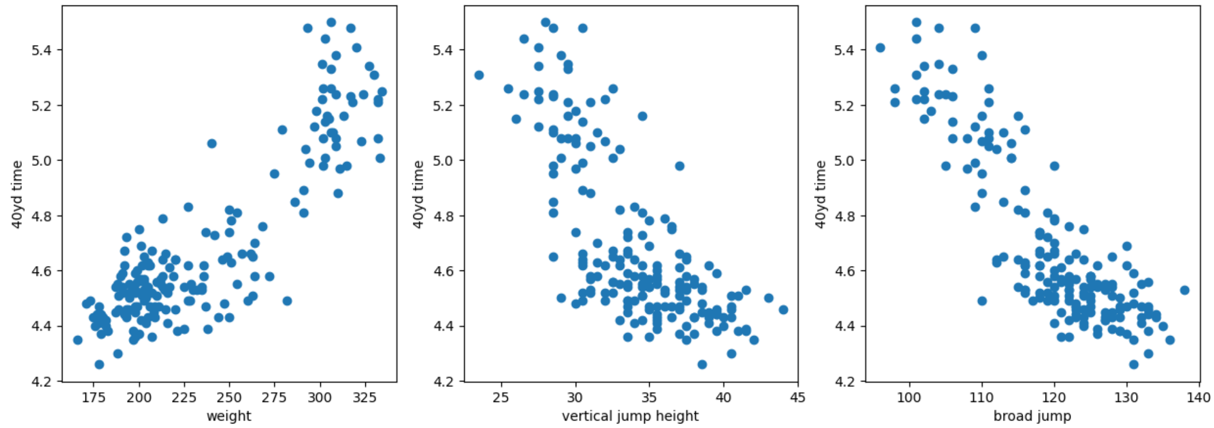
Out[24]:

	Player	Wt	40yd	Vertical	Broad Jump
1	Yasir Abdullah	237	4.47	36.5	129.0
3	Jordan Addison	173	4.49	34.0	122.0
4	Adetomiwa Adebawore	282	4.49	37.5	125.0
8	Jake Andrews	305	5.15	26.0	102.0
10	Malaesala Aumavae-Laulu	317	5.23	28.5	106.0
...	...	...	...	...	...
307	Michael Wilson	213	4.58	37.5	125.0
309	Dee Winters	227	4.49	30.5	117.0
312	Darnell Wright	333	5.01	29.0	114.0
314	Luke Wypler	303	5.14	30.5	106.0
317	Byron Young	250	4.43	38.0	132.0

182 rows × 5 columns

In [25]:
```
# scatter plots of individual predictors vs response
fig_1, axes = subplots(nrows=1, ncols=3, figsize=(15,5))

# 40yd vs weight
axes[0].scatter(combine[['Wt']], combine[['40yd']], marker='o')
axes[0].set_xlabel("weight")
axes[0].set_ylabel("40yd time")

# 40yd vs vertical
axes[1].scatter(combine[['Vertical']], combine[['40yd']], marker='o')
axes[1].set_xlabel("vertical jump height")
axes[1].set_ylabel("40yd time")

# 40yd vs broad jump
axes[2].scatter(combine[['Broad Jump']], combine[['40yd']], marker='o')
axes[2].set_xlabel("broad jump")
axes[2].set_ylabel("40yd time")
```

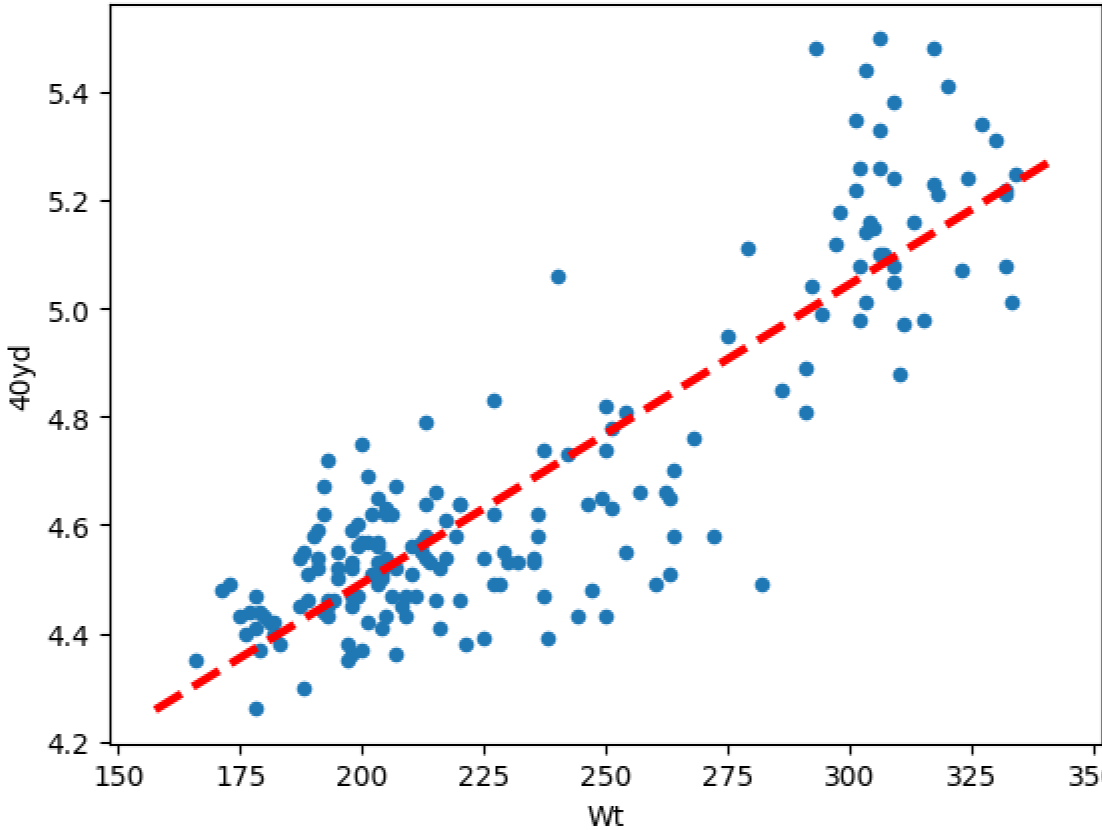
Out[25]:
```
Text(0, 0.5, '40yd time')
```



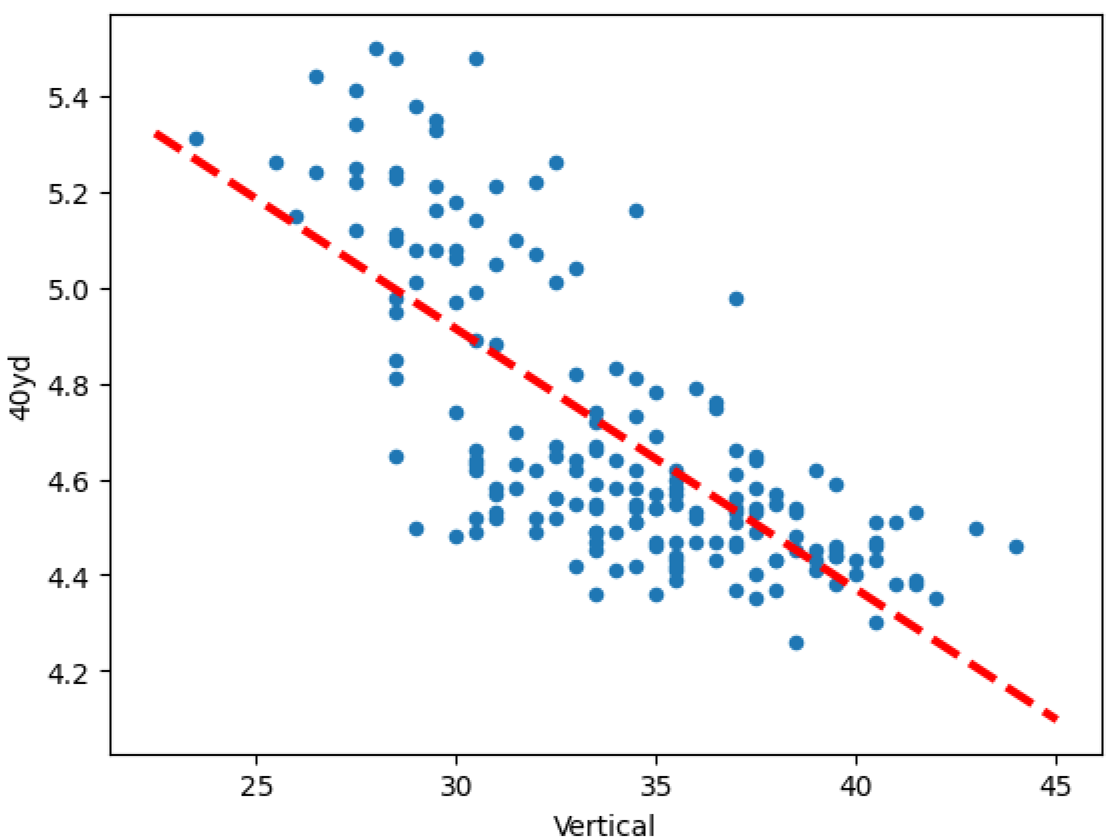
In [26]:
```
# we regress 40 yd time onto each of weight, vertical jump and broad jump
Y = combine['40yd']

# 40yd regressed on weight
X_1 = pd.DataFrame({'intercept': np.ones(combine.shape[0]), 'weight': combir
model_1 = sm.OLS(Y, X_1)
results_1 = model_1.fit()

# 40yd regressed on vertical jump
X_2 = pd.DataFrame({'intercept': np.ones(combine.shape[0]), 'vertical jump':
model_2 = sm.OLS(Y, X_2)
results_2 = model_2.fit()

# 40yd regressed on broad jump
X_3 = pd.DataFrame({'intercept': np.ones(combine.shape[0]), 'broad jump': cc
model_3 = sm.OLS(Y, X_3)
results_3 = model_3.fit()
```

In [27]:
```
# function that plots an abline line
def abline(ax, b, m, *args, **kwargs):
    "Add a line with slope m and intercept b to ax"
    xlim = ax.get_xlim()
    ylim = [m * xlim[0] + b, m * xlim[1] + b]
    ax.plot(xlim, ylim, *args, **kwargs)
```

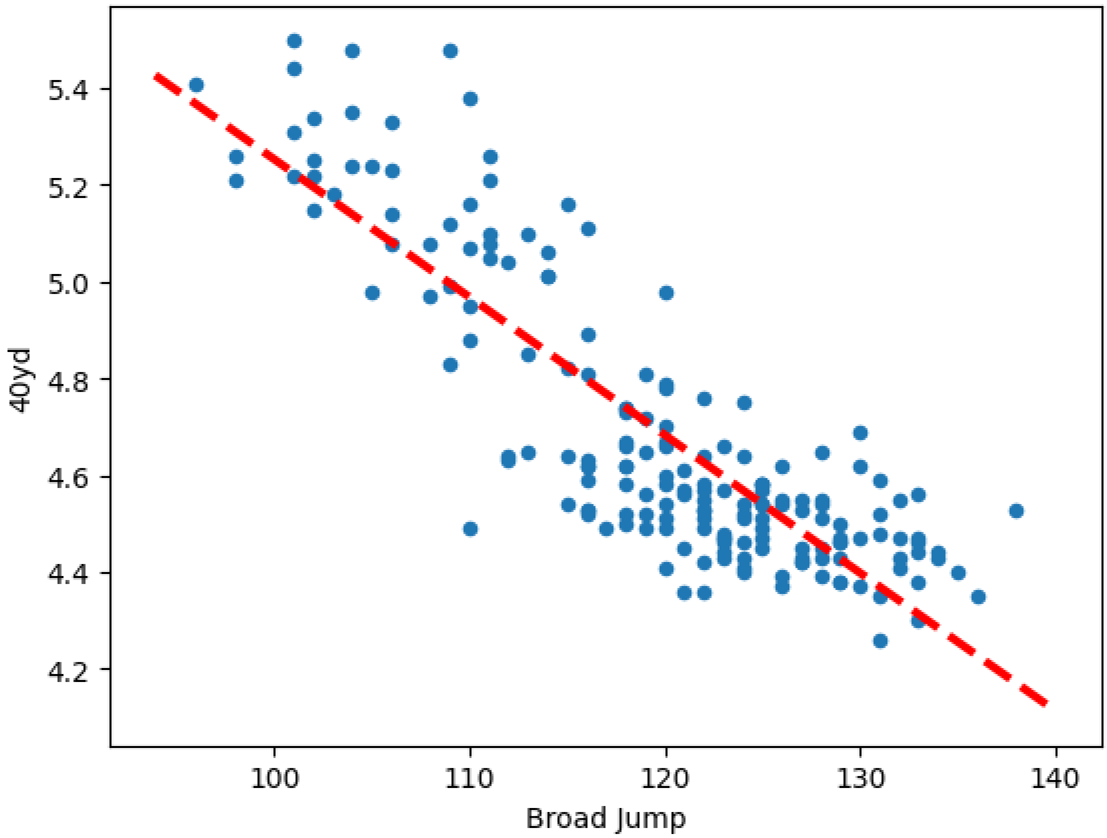
In [28]:
```
# plotting the least squares regression line with the scatter plot of weight
ax_1 = combine.plot.scatter('Wt', '40yd')
abline(ax_1, results_1.params[0], results_1.params[1], 'r--', linewidth=3)
```



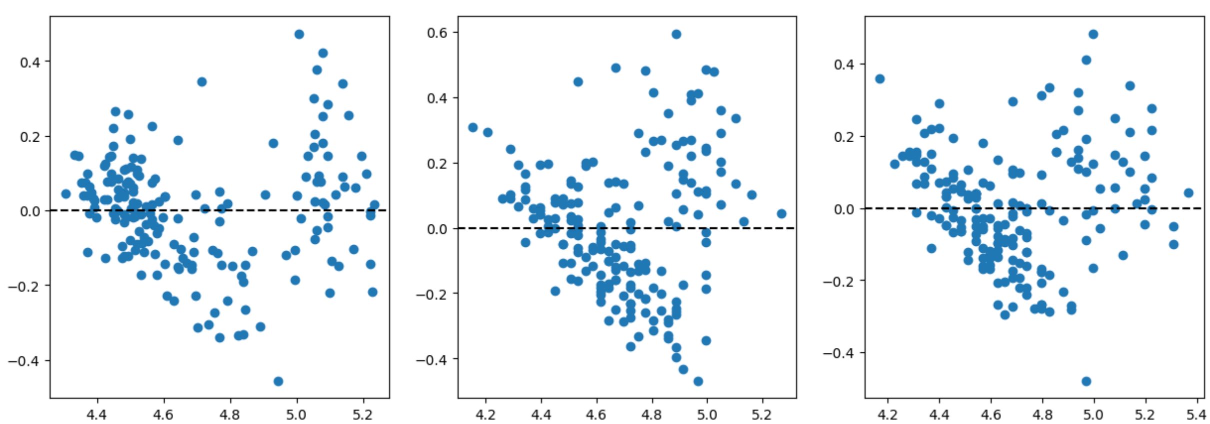
In [29]:
```
# plotting the least squares regression line with the scatter plot of vertic
ax_2 = combine.plot.scatter('Vertical', '40yd')
abline(ax_2, results_2.params[0], results_2.params[1], 'r--', linewidth=3)
```



In [30]:
```
# plotting the least squares regression line with the scatter plot of weight
ax_3 = combine.plot.scatter('Broad Jump', '40yd')
abline(ax_3, results_3.params[0], results_3.params[1], 'r--', linewidth=3)
```

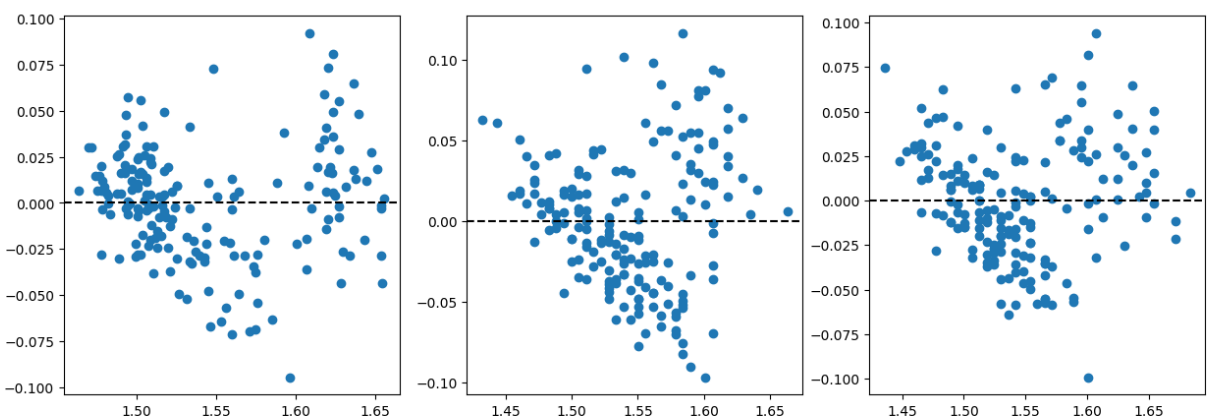


In [31]:
```
# we plot the fitted values vs residuals on one figure
fig_2, axes_2 = subplots(nrows=1,ncols=3,figsize=(15,5))

axes_2[0].scatter(results_1.fittedvalues, results_1.resid)
axes_2[1].scatter(results_2.fittedvalues, results_2.resid)
axes_2[2].scatter(results_3.fittedvalues, results_3.resid)

# placing horizontal lines at residual value = 0

axes_2[0].axhline(0,c='k', ls='--');
axes_2[1].axhline(0,c='k', ls='--');
axes_2[2].axhline(0,c='k', ls='--');
```



In [32]:
```
# we detect non-constant variance or heteroskedasticity in the residual vs f
# we therefore apply a concave function to the response

log_Y = np.log(Y)
```

In [33]:
```
# we once again fit the simple linear regression line
model_1_log = sm.OLS(log_Y, X_1)
results_1_log = model_1_log.fit()

model_2_log = sm.OLS(log_Y, X_2)
results_2_log = model_2_log.fit()

model_3_log = sm.OLS(log_Y, X_3)
results_3_log = model_3_log.fit()

# we plot the fitted values vs residuals of the log response on one figure
fig_2, axes_2 = subplots(nrows=1,ncols=3,figsize=(15,5))

axes_2[0].scatter(results_1_log.fittedvalues, results_1_log.resid)
axes_2[1].scatter(results_2_log.fittedvalues, results_2_log.resid)
axes_2[2].scatter(results_3_log.fittedvalues, results_3_log.resid)

# placing horizontal lines at residual value = 0

axes_2[0].axhline(0,c='k', ls='--');
axes_2[1].axhline(0,c='k', ls='--');
axes_2[2].axhline(0,c='k', ls='--');
```



In [34]:
```
# we notice no significant change in heteroskedasticity so we can continue w

# we wish to construct a multiple linear regression model with polynomial te
# to account for the non-linearity and collinearity of the features

X_mlr1 = MS([poly('Broad Jump', degree=2), poly('Wt', degree=2), ('Vertical'
X_mlr2 = X_mlr1.transform(combine)
M = sm.OLS(Y, X_mlr1.transform(combine)).fit()

# retrieve summary of coefficients
summarize(M)
```

Out[34]:

	coef	std err	t	P>|t|
intercept	4.823400	0.138000	35.034	0.000
poly(Broad Jump, degree=2)[0]	-1.511200	0.371000	-4.068	0.000
poly(Broad Jump, degree=2)[1]	0.462100	0.141000	3.287	0.001
poly(Wt, degree=2)[0]	2.050400	0.206000	9.955	0.000
poly(Wt, degree=2)[1]	0.336700	0.135000	2.485	0.014
Vertical:Broad Jump	-0.000032	0.000034	-0.943	0.347

In [35]:
```
# load test data
test_combine = pd.read_csv('test_set_combine2022.csv')

# display test data
test_combine

# restrict to desired columns
test_combine = test_combine[['Player', 'Wt', '40yd', 'Vertical', 'Broad Jump

# remove observations with non-values for any of 40yd, weight, vertical or b
test_combine = test_combine.dropna()

# view table of data
test_combine

# create table of only predictors
X_test = test_combine[['Wt', 'Vertical', 'Broad Jump']]

# transform data
X_test_transform = X_mlr1.transform(X_test)

# get predictions
preds = M.get_prediction(X_test_transform)
```

In [36]:
```
# get predictions as a table
forty_predicted_times = pd.DataFrame({'40yd prediction': preds.predicted_mea

# display results
forty_predicted_times
```

Out[36]:

	40yd prediction
0	4.852941
1	4.668249
2	4.601060
3	4.569997
4	4.526380
...	...
182	4.445948
183	4.502931
184	5.033012
185	5.111195
186	4.761855

187 rows × 1 columns

In [37]:

```
# display results of actual 2022 combine results
test_combine
```

Out[37]:

	Player	Wt	40yd	Vertical	Broad Jump
0	Cal Adomitis	235	4.97	29.5	107.0
1	Austin Allen	253	4.83	34.0	121.0
4	Tyler Allgeier	224	4.60	33.0	120.0
5	Troy Andersen	243	4.42	36.0	128.0
6	Tycen Anderson	209	4.36	35.5	123.0
...	...	...	...	...	...
314	JT Woods	195	4.36	39.5	128.0
315	Mike Woods	204	4.55	34.5	125.0
319	Devonte Wyatt	304	4.77	29.0	111.0
322	Nick Zakelj	316	5.13	28.5	110.0
323	Bailey Zappe	215	4.88	30.0	109.0

187 rows × 5 columns

In [52]:

```
# we iterate through the n observations to compute the test MSE
total = 0

for observed_value, predicted_value in zip(test_combine['40yd'], preds.predi
    total += (observed_value - predicted_value)**2
```

```
n = test_combine.shape[0]

test_MSE = (1/n)*(total)
```

In [53]:
```
test_MSE
```

Out[53]: 0.014451420876737892

In [63]:
```
# find the average of the test data
avg_time = test_combine['40yd'].sum()
avg_time = (1/n)*(avg_time)

# create an array of avg 40 time
avg_time_array = np.ones(test_combine.shape[0]) * avg_time

# compute baseline MSE
total_2 = 0

for ob_value, mean_value in zip(test_combine['40yd'], avg_time_array):
    total_2 += (ob_value - mean_value)**2

baseline_MSE = (1/n)*(total_2)
```

In [64]:
```
baseline_MSE
```

Out[64]: 0.09316607852669506

We observe a 84.5% decrease in MSE when comparing the baseline MSE to our test MSE. We can conclude our model is an accurate predictor of forty-yard dash times.