In [2]:
```python
'''
Regina Catipon
MACS 30100
2/6/2020
'''
import numpy as np
import random
import math
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
import itertools
import statsmodels.api as sm
```

Homework 3: Linear Model Selection and Regularization

1. (5 points) Generate a data set with p = 20 features, n = 1000 observations, and an associated quantitative response vector generated according to the model Y = Xβ + episilon where β has some elements that are exactly equal to zero

In [3]:
```python
random.seed(123)

x = np.random.uniform(size=(1000, 20))

x = pd.DataFrame(x)
```

In [4]:
```python
x.head()
```

Out[4]:

	0	1	2	3	4	5	6	7	8	
0	0.052957	0.667867	0.844883	0.900268	0.081392	0.744863	0.899439	0.879189	0.986715	0.4
1	0.125246	0.838739	0.573174	0.303372	0.262918	0.336520	0.789148	0.962172	0.742410	0.5
2	0.525230	0.516743	0.978712	0.654858	0.327557	0.820102	0.123899	0.026516	0.127198	0.9
3	0.643179	0.289518	0.788026	0.396482	0.388104	0.699363	0.541265	0.874668	0.588942	0.5
4	0.296170	0.301884	0.646417	0.006882	0.296197	0.580819	0.588030	0.340039	0.971965	0.1

In [5]:
```python
B = np.random.uniform(size=(20))
B[0] = 0
B[5] = 0
B[18] = 0
B = B.T
B = pd.DataFrame(B)
```

In [6]:
```python
epsilon = np.random.uniform(0, .5, 1000)
epsilon = pd.DataFrame(epsilon)
```

In [7]:
```
y = x.dot(B) + epsilon
y = pd.DataFrame(y)
```

2. (10 points) Split your data set into a training set containing 100 observations and a test set containing 900 observations.

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = .9
)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(100, 20) (900, 20) (100, 1) (900, 1)
```

3. (10 points) Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size. For which model size does the training set MSE take on its minimum value?

In [9]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [10]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifi
er
from sklearn.metrics import mean_squared_error
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.classifier import LogisticRegression
```

In [11]:
```
## Step Forward Feature Selection (SFS)
```

In [12]:
```
from sklearn.linear_model import LinearRegression
# will find the maximally performing subset of the 20 features
sfs = SFS(LinearRegression(),
         k_features = 20,
         forward = True,
         floating = False,
         verbose = 2,
         scoring = "neg_mean_squared_error",
         ).fit(X_train, y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    0.3s finished

[2020-02-09 19:11:49] Features: 1/20 -- score: -0.5779627816173483[Para
llel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  19 out of  19 | elapsed:    0.3s finished

[2020-02-09 19:11:49] Features: 2/20 -- score: -0.4919498400423567[Para
llel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed:    0.3s finished

[2020-02-09 19:11:50] Features: 3/20 -- score: -0.407415360080571[Paral
lel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  17 out of  17 | elapsed:    0.2s finished

[2020-02-09 19:11:50] Features: 4/20 -- score: -0.3343477998393916[Para
llel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:    0.3s finished

[2020-02-09 19:11:50] Features: 5/20 -- score: -0.2748266473761724[Para
llel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:    0.2s finished

[2020-02-09 19:11:50] Features: 6/20 -- score: -0.24136435079336174[Par
allel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    0.2s finished

[2020-02-09 19:11:50] Features: 7/20 -- score: -0.20219791014408414[Par
allel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    0.3s finished

[2020-02-09 19:11:51] Features: 8/20 -- score: -0.17668606577850787[Par
allel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
```

```
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    0.2s finished

[2020-02-09 19:11:51] Features: 9/20 -- score: -0.1443574913180436[Para
llel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    0.2s finished

[2020-02-09 19:11:51] Features: 10/20 -- score: -0.11378751438584737[Pa
rallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.1s finished

[2020-02-09 19:11:51] Features: 11/20 -- score: -0.09831985732516821[Pa
rallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.2s finished

[2020-02-09 19:11:51] Features: 12/20 -- score: -0.08643859252282854[Pa
rallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.1s finished

[2020-02-09 19:11:52] Features: 13/20 -- score: -0.06813338103903073[Pa
rallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.1s finished

[2020-02-09 19:11:52] Features: 14/20 -- score: -0.05536952609085103[Pa
rallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.1s finished

[2020-02-09 19:11:52] Features: 15/20 -- score: -0.045248030504901594[P
arallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.1s finished

[2020-02-09 19:11:52] Features: 16/20 -- score: -0.03895412160877911[Pa
rallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s finished

[2020-02-09 19:11:52] Features: 17/20 -- score: -0.03132111659185556[Pa
rallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s finished

[2020-02-09 19:11:52] Features: 18/20 -- score: -0.031798543026774075[P
arallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s finished

[2020-02-09 19:11:52] Features: 19/20 -- score: -0.03240447684369039[Pa
rallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished

[2020-02-09 19:11:52] Features: 20/20 -- score: -0.034194772969964114
```

In [40]:
```
sfs.subsets_
```

```
Out[40]: {1: {'feature_idx': (9,),
           'cv_scores': array([-0.94990298, -0.46140848, -0.50471063, -0.4015368
         8, -0.57225494]),
           'avg_score': -0.5779627816173483,
           'feature_names': (9,)},
          2: {'feature_idx': (9, 12),
           'cv_scores': array([-0.71021179, -0.3457341 , -0.44253918, -0.4427721
         8, -0.51849194]),
           'avg_score': -0.4919498400423567,
           'feature_names': (9, 12)},
          3: {'feature_idx': (9, 12, 16),
           'cv_scores': array([-0.5456009 , -0.27658113, -0.50269489, -0.3418881
         3, -0.37031176]),
           'avg_score': -0.407415360080571,
           'feature_names': (9, 12, 16)},
          4: {'feature_idx': (3, 9, 12, 16),
           'cv_scores': array([-0.40886273, -0.30776477, -0.37998361, -0.2296861
         9, -0.3454417 ]),
           'avg_score': -0.3343477998393916,
           'feature_names': (3, 9, 12, 16)},
          5: {'feature_idx': (3, 9, 12, 14, 16),
           'cv_scores': array([-0.29327866, -0.27831222, -0.3519868 , -0.2451598
         8, -0.20539568]),
           'avg_score': -0.2748266473761724,
           'feature_names': (3, 9, 12, 14, 16)},
          6: {'feature_idx': (3, 9, 12, 14, 15, 16),
           'cv_scores': array([-0.25274435, -0.24292806, -0.34073092, -0.2473720
         6, -0.12304637]),
           'avg_score': -0.24136435079336174,
           'feature_names': (3, 9, 12, 14, 15, 16)},
          7: {'feature_idx': (3, 4, 9, 12, 14, 15, 16),
           'cv_scores': array([-0.17799952, -0.21199618, -0.25314416, -0.2543814
         8, -0.11346822]),
           'avg_score': -0.20219791014408414,
           'feature_names': (3, 4, 9, 12, 14, 15, 16)},
          8: {'feature_idx': (3, 4, 9, 11, 12, 14, 15, 16),
           'cv_scores': array([-0.16751848, -0.23903963, -0.19577643, -0.1941437
         9, -0.086952  ]),
           'avg_score': -0.17668606577850787,
           'feature_names': (3, 4, 9, 11, 12, 14, 15, 16)},
          9: {'feature_idx': (2, 3, 4, 9, 11, 12, 14, 15, 16),
           'cv_scores': array([-0.16489431, -0.1600066 , -0.12574652, -0.1549908
         2, -0.11614921]),
           'avg_score': -0.1443574913180436,
           'feature_names': (2, 3, 4, 9, 11, 12, 14, 15, 16)},
          10: {'feature_idx': (2, 3, 4, 9, 11, 12, 14, 15, 16, 17),
           'cv_scores': array([-0.17181918, -0.10776434, -0.07575512, -0.1237630
         4, -0.08983589]),
           'avg_score': -0.11378751438584737,
           'feature_names': (2, 3, 4, 9, 11, 12, 14, 15, 16, 17)},
          11: {'feature_idx': (2, 3, 4, 9, 11, 12, 13, 14, 15, 16, 17),
           'cv_scores': array([-0.14100403, -0.119931  , -0.06754543, -0.1010217
         6, -0.06209707]),
           'avg_score': -0.09831985732516821,
           'feature_names': (2, 3, 4, 9, 11, 12, 13, 14, 15, 16, 17)},
          12: {'feature_idx': (1, 2, 3, 4, 9, 11, 12, 13, 14, 15, 16, 17),
           'cv_scores': array([-0.10561162, -0.11714602, -0.06614297, -0.0703204
```

```
5, -0.07297189]),
  'avg_score': -0.08643859252282854,
  'feature_names': (1, 2, 3, 4, 9, 11, 12, 13, 14, 15, 16, 17)},
 13: {'feature_idx': (1, 2, 3, 4, 8, 9, 11, 12, 13, 14, 15, 16, 17),
  'cv_scores': array([-0.07169159, -0.09544536, -0.05435884, -0.0488810
8, -0.07029004]),
  'avg_score': -0.06813338103903073,
  'feature_names': (1, 2, 3, 4, 8, 9, 11, 12, 13, 14, 15, 16, 17)},
 14: {'feature_idx': (1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 14, 15, 16, 17),
  'cv_scores': array([-0.04986344, -0.08657075, -0.05773532, -0.0334350
3, -0.04924309]),
  'avg_score': -0.05536952609085103,
  'feature_names': (1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 14, 15, 16, 17)},
 15: {'feature_idx': (1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17),
  'cv_scores': array([-0.03082227, -0.06971054, -0.04964391, -0.0373734
4, -0.03868998]),
  'avg_score': -0.045248030504901594,
  'feature_names': (1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
7)},
 16: {'feature_idx': (1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1
6, 17),
  'cv_scores': array([-0.02678918, -0.05439055, -0.03597038, -0.0256060
1, -0.05201449]),
  'avg_score': -0.03895412160877911,
  'feature_names': (1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17)},
 17: {'feature_idx': (1,
   2,
   3,
   4,
   6,
   7,
   8,
   9,
   10,
   11,
   12,
   13,
   14,
   15,
   16,
   17,
   19),
  'cv_scores': array([-0.02045314, -0.03564658, -0.03414136, -0.0245928
5, -0.04177165]),
  'avg_score': -0.03132111659185556,
  'feature_names': (1,
   2,
   3,
   4,
   6,
   7,
   8,
   9,
   10,
   11,
```

```
    12,
    13,
    14,
    15,
    16,
    17,
    19)},
  18: {'feature_idx': (1,
    2,
    3,
    4,
    6,
    7,
    8,
    9,
    10,
    11,
    12,
    13,
    14,
    15,
    16,
    17,
    18,
    19),
   'cv_scores': array([-0.02061282, -0.0374657 , -0.0346752 , -0.0245042
8, -0.04173472]),
   'avg_score': -0.031798543026774075,
   'feature_names': (1,
    2,
    3,
    4,
    6,
    7,
    8,
    9,
    10,
    11,
    12,
    13,
    14,
    15,
    16,
    17,
    18,
    19)},
  19: {'feature_idx': (0,
    1,
    2,
    3,
    4,
    6,
    7,
    8,
    9,
    10,
    11,
```

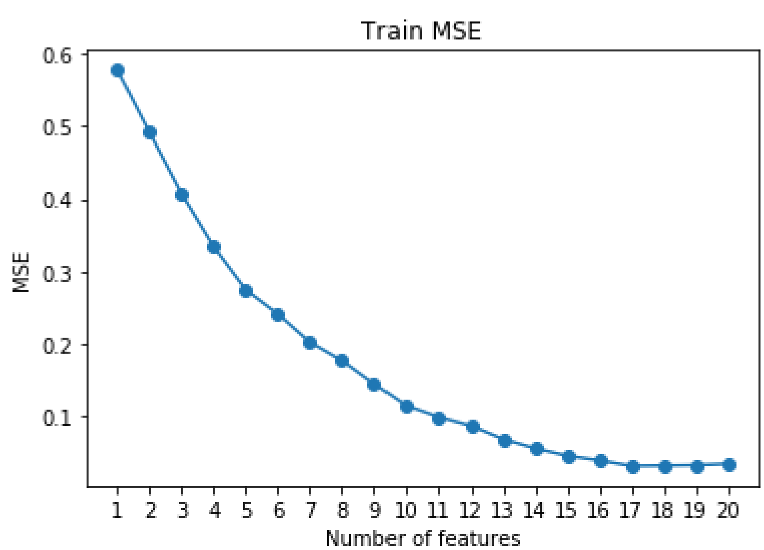
```
    12,
    13,
    14,
    15,
    16,
    17,
    18,
    19),
   'cv_scores': array([-0.02085856, -0.03901126, -0.03502147, -0.0247161
6, -0.04241494]),
   'avg_score': -0.03240447684369039,
   'feature_names': (0,
    1,
    2,
    3,
    4,
    6,
    7,
    8,
    9,
    10,
    11,
    12,
    13,
    14,
    15,
    16,
    17,
    18,
    19)},
  20: {'feature_idx': (0,
    1,
    2,
    3,
    4,
    5,
    6,
    7,
    8,
    9,
    10,
    11,
    12,
    13,
    14,
    15,
    16,
    17,
    18,
    19),
   'cv_scores': array([-0.02057508, -0.03833289, -0.04162696, -0.0248012
6, -0.04563768]),
   'avg_score': -0.034194772969964114,
   'feature_names': (0,
    1,
    2,
    3,
```

```
         4,
         5,
         6,
         7,
         8,
         9,
         10,
         11,
         12,
         13,
         14,
         15,
         16,
         17,
         18,
         19)}}
```

In [14]:
```
train_MSE = []
for n_size in range(1,21):
    train_MSE.append(-1 * sfs.subsets_[n_size]['avg_score'])
```

In [48]:
```
# Plot the test set MSE associated with the best model of each size.
plt.plot(range(1, 21), train_MSE, label='train MSE', marker='o')
plt.title('Train MSE')
plt.xlabel('Number of features')
plt.ylabel('MSE')
plt.xticks(range(1, 21))
plt
```

Out[48]:
```
<module 'matplotlib.pyplot' from '/Users/reginacatipon/anaconda3/lib/py
thon3.7/site-packages/matplotlib/pyplot.py'>
```



The number of features that produced the lowest train MSE is 17 at a rate of 0.031. However from 16 to 20, the difference between accuracy and the number of predictors becomes slight. It makes that the lowest training MSE is at 17 predictors because the original dataset had three Betas that were 0. The test MSE should match the train MSE.

4. (5 points) Plot the test set MSE associated with the best model of each size.

In [16]:
```
from sklearn.metrics import mean_squared_error as MSE
```

In [17]:
```
X_train[[1,0]]
```

Out[17]:

	1	0
313	0.251328	0.720689
136	0.029880	0.027155
889	0.712399	0.548166
155	0.264510	0.371877
366	0.964868	0.023894
...	...	...
128	0.577090	0.825944
623	0.742800	0.432118
589	0.463168	0.393496
846	0.872824	0.699601
888	0.283461	0.472505

100 rows × 2 columns

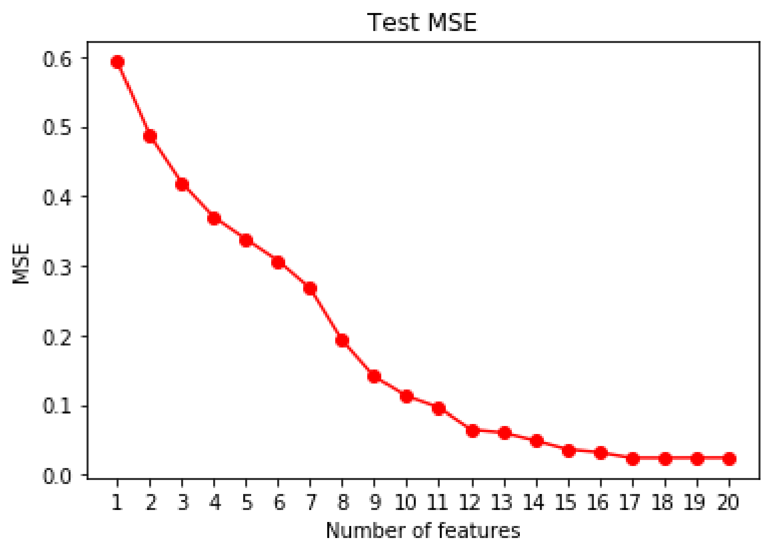
In [18]:
```
test_MSE = []
for i in range(1,21):
    ols_i = LinearRegression().fit(X_train[list(sfs.subsets_[i]['feature_idx'])], y_train)
    prediction_i = ols_i.predict(X_test[list(sfs.subsets_[i]['feature_idx'])])
    mse_i = MSE(prediction_i, y_test)
    test_MSE.append(mse_i)

test_MSE
```

Out[18]:
```
[0.5928550751349376,
 0.4876983886012735,
 0.41909504692808947,
 0.3698888409050719,
 0.33813004931481766,
 0.3071528597399765,
 0.2676095280021955,
 0.1926864976930774,
 0.14052075112109996,
 0.11247317429804322,
 0.09613525869923614,
 0.06491050462734081,
 0.060358517293477576,
 0.04911046245036431,
 0.03660268826045491,
 0.03211008085757004,
 0.024146929974764433,
 0.0241232910675067,
 0.024241251361123414,
 0.02426366408733416]
```

In [49]:
```
plt.plot(range(1, 21), test_MSE, label='test MSE', marker='o', color="re
d")
plt.title('Test MSE')
plt.xlabel('Number of features')
plt.ylabel('MSE')
plt.xticks(range(1, 21))
plt
```

Out[49]:
```
<module 'matplotlib.pyplot' from '/Users/reginacatipon/anaconda3/lib/py
thon3.7/site-packages/matplotlib/pyplot.py'>
```



5. (5 points) For which model size does the test set MSE take on its minimum value? Comment on your results.

From the plot and from the list of MSEs, we see that the lowest MSE value was for 17 features. It is not surprising that n feature size of 17 returns the lowest test MSE because in the generated data, I inputed 0 for three Betas. To verify, I checked the dictionary of subsets_, and the excluded predictors at 17 features were, in fact, the predictors p0, p5, and p18-- the predictors where I placed the zero Betas.

6. (10 points) How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient sizes.

In [41]:
```
print(sfs.subsets_[17]['feature_idx'])
```

```
(1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19)
```

In [42]:
```
ols_subset = LinearRegression().fit(X_train[list(sfs.subsets_[17]['featu
re_idx'])], y_train)
ols_subset.coef_
```

Out[42]:
```
array([[0.50263147, 0.70897386, 0.70821929, 0.6973405 , 0.32812756,
        0.26430695, 0.40481773, 1.03601358, 0.2931415 , 0.90330532,
        0.93692642, 0.45128996, 0.87383514, 0.7379998 , 0.8237134 ,
        0.48474351, 0.27533289]])
```

In [43]:
```
comparison = pd.DataFrame(B.iloc[[1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13
, 14, 15, 16, 17, 19],0])
comparison = comparison.rename(columns={0:"Betas"})
comparison = comparison.reset_index()
comparison = comparison.rename(columns={"index":"Feature"})
coeffs = pd.DataFrame(ols_subset.coef_).T
comparison['Coefficients'] = coeffs
```

In [44]:
```
# comment on the coefficient size
comparison['Abs Diff'] = abs(comparison["Betas"] - comparison['Coefficie
nts'])
comparison
```

Out[44]:

	Feature	Betas	Coefficients	Abs Diff
0	1	0.533082	0.502631	0.030451
1	2	0.718205	0.708974	0.009231
2	3	0.760187	0.708219	0.051968
3	4	0.707473	0.697340	0.010133
4	6	0.347550	0.328128	0.019423
5	7	0.234877	0.264307	0.029430
6	8	0.292266	0.404818	0.112551
7	9	0.995734	1.036014	0.040280
8	10	0.320082	0.293142	0.026941
9	11	0.992121	0.903305	0.088816
10	12	0.969101	0.936926	0.032175
11	13	0.408766	0.451290	0.042524
12	14	0.781552	0.873835	0.092283
13	15	0.698774	0.738000	0.039226
14	16	0.864112	0.823713	0.040398
15	17	0.510617	0.484744	0.025873
16	19	0.276595	0.275333	0.001262

In [24]:
```
comparison['Abs Diff'].sum()/13
```

Out[24]: 0.2441009104029323

To compare the coefficient sizes, I used the absolute value of the difference and I found that predictor 19 had the smallest absolute difference between the true Betas and the generated coefficient. Whereas predictor 6 had the highest absolute difference.

7. (10 points) Create a plot displaying

$$\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_j^r)^2}$$

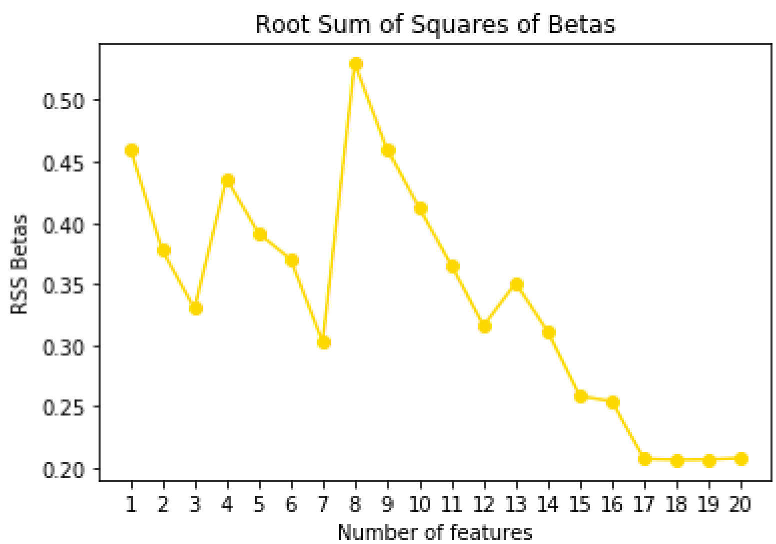
for a range of values of r, where $\hat{\beta}_j^r$ is the jth coefficient estimate for the best model containing r coefficients. Comment on what you observe. How does this compare to the test MSE plot?

In [25]:
```
features = []
rss_betas = []
for i in range(20):
    for ind in sfs.subsets_[i + 1]['feature_idx']:
        if ind not in features:
            features.append(ind)
    ols_i = LinearRegression().fit(X_train[features], y_train)
    actual_b = [b[0] for b in np.array(B)[features]]
    diff_beta = ols_i.coef_[0] - np.array(actual_b)
    sq_beta = diff_beta**2
    rss_i = sq_beta.sum()**0.5
    rss_betas.append(rss_i)
rss_betas
```

Out[25]:
```
[0.4605196050260185,
 0.37741044080469766,
 0.33014855033252216,
 0.436064961922642,
 0.3914260096176989,
 0.37021761093727507,
 0.3027363190335416,
 0.5297532309752305,
 0.4600820385033422,
 0.41174417754264836,
 0.3649093346257824,
 0.3159635883871001,
 0.3509123043503953,
 0.31114833980058065,
 0.25855346511350136,
 0.2542525543144556,
 0.20776173412526755,
 0.20683000240102664,
 0.20707289288321024,
 0.2082886180590684]
```

In [52]:
```
plt.plot(range(1, 21), rss_betas, label='rss_betas', marker = "o", color
="gold")
plt.title('Root Sum of Squares of Betas')
plt.xlabel('Number of features')
plt.ylabel('RSS Betas')
plt.xticks(range(1, 21))
plt
```

Out[52]:
```
<module 'matplotlib.pyplot' from '/Users/reginacatipon/anaconda3/lib/py
thon3.7/site-packages/matplotlib/pyplot.py'>
```



Compared to the test MSE plot, we again see unique behavior at the n = 17 features point. There are a few local maximum where the RSS jumps up, but as the number of features approach the actual number of predictors, we see more expected behavior. As the graph approaches n = 17, we see a decreases in difference between Betas.

2. Application Exercises

In [27]:
```
from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV
from sklearn.model_selection import cross_validate
```

In [28]:
```
# read in data
train_data = pd.read_csv('./data/gss_train.csv')
test_data = pd.read_csv('./data/gss_test.csv')
```

In [71]:
```
# preview data
train_data.columns
```

Out[71]:
```
Index(['age', 'attend', 'authoritarianism', 'black', 'born', 'childs',
       'colath', 'colrac', 'colcom', 'colmil', 'colhomo', 'colmslm',
       'con_govt', 'egalit_scale', 'evangelical', 'grass', 'happy',
       'hispanic_2', 'homosex', 'income06', 'mode', 'owngun', 'polview
s',
       'pornlaw2', 'pray', 'pres08', 'reborn_r', 'science_quiz', 'sex',
'sibs',
       'social_connect', 'south', 'teensex', 'tolerance', 'tvhours',
       'vetyears', 'wordsum', 'degree_HS', 'degree_Junior.Coll',
       'degree_Bachelor.deg', 'degree_Graduate.deg', 'marital_Widowed',
       'marital_Divorced', 'marital_Separated', 'marital_Never.marrie
d',
       'news_FEW.TIMES.A.WEEK', 'news_ONCE.A.WEEK', 'news_LESS.THAN.ONC
E.WK',
       'news_NEVER', 'partyid_3_Ind', 'partyid_3_Rep', 'relig_CATHOLI
C',
       'relig_JEWISH', 'relig_NONE', 'relig_OTHER', 'relig_BUDDHISM',
       'relig_HINDUISM', 'relig_OTHER.EASTERN', 'relig_MOSLEM.ISLAM',
       'relig_ORTHODOX.CHRISTIAN', 'relig_CHRISTIAN', 'relig_NATIVE.AME
RICAN',
       'relig_INTER.NONDENOMINATIONAL', 'social_cons3_Mod',
       'social_cons3_Conserv', 'spend3_Mod', 'spend3_Liberal', 'zodiac_
TAURUS',
       'zodiac_GEMINI', 'zodiac_CANCER', 'zodiac_LEO', 'zodiac_VIRGO',
       'zodiac_LIBRA', 'zodiac_SCORPIO', 'zodiac_SAGITTARIUS',
       'zodiac_CAPRICORN', 'zodiac_AQUARIUS', 'zodiac_PISCES'],
      dtype='object')
```

In [31]:
```
#split, train and test
Y_test, x_test = test_data['egalit_scale'], test_data[test_data.columns.
difference(['egalit_scale'])]
Y_train, x_train = train_data['egalit_scale'], train_data[train_data.col
umns.difference(['egalit_scale'])]
```

In [36]:
```
# print shapes
print("Y_train", Y_train.shape)
print("Y_test", Y_test.shape)
print("x_train", x_train.shape)
print("x_test", x_test.shape)
```

```
Y_train (1481,)
Y_test (493,)
x_train (1481, 77)
x_test (493, 77)
```

1. (10 points) Fit a least squares linear model on the training set, and report the test MSE.

In [82]:
```
#alphas are lambdas in sklearn

iterations = 10000

ols = LinearRegression().fit(x_train, Y_train)

ols_error_train = MSE(Y_train, ols.predict(x_train))
ols_error_test = MSE(Y_test, ols.predict(x_test))
```

In [83]:
```
print('####')
print('Test error')
print(ols_error_train)

print('Test error')
print(ols_error_test)
print('####')
```

```
####
Test error
55.12263854924573
Test error
63.21362962301501
####
```

2. (10 points) Fit a ridge regression model on the training set, with λ chosen by 10-fold cross-validation. Report the test MSE. (below)

In [84]:
```
# define function to set lambda
def get_lambda_MSE(modelCV):

    modelCV.fit(x_train, Y_train)

    alpha = modelCV.alpha_

    train_mse = MSE(Y_train, modelCV.predict(x_train))

    test_mse = MSE(Y_test, modelCV.predict(x_test))

    return alpha, train_mse, test_mse, modelCV

n = x_train.shape[1]
```

In [105]:
```
#Fit a ridge regression model on the training set
rdg =  RidgeCV(alphas = np.linspace(0.1, 2 ,10))
rdg_alpha, rdg_error_train, rdg_error_test, rdg_model = get_lambda_MSE(r
dg)
coefficients = sum(np.abs(rdg_model.coef_) < 0.1)
```

In [170]:
```
print('####')
print('Ridge Regression')
print(coefficients, 'coefficients are non-zero estimates')
print('####')
```

```
####
Ridge Regression
9 coefficients are non-zero estimates
####
```

3. (10 points) Fit a lasso regression on the training set, with λ chosen by 10-fold cross-validation. Report the test MSE, along with the number of non-zero coefficient estimates.(below)

In [108]:
```
# lasso regression
las = LassoCV(n_alphas=10, max_iter=iterations)
las_alpha, las_error_train, las_error_test, las_model = get_lambda_MSE(l
as)
coefficients_lasso = sum(np.abs(las_model.coef_) < 0.1)
```

```
/Users/reginacatipon/anaconda3/lib/python3.7/site-packages/sklearn/mode
l_selection/_split.py:1978: FutureWarning: The default value of cv will
change from 3 to 5 in version 0.22. Specify it explicitly to silence th
is warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

In [171]:
```
print('####')
print('Lasso Regression')
print(coefficients_lasso, 'coefficients are non-zero estimates')
print('####')
```

```
####
Lasso Regression
59 coefficients are non-zero estimates
####
```

4. (10 points) Fit an elastic net regression model on the training set, with α and λ chosen by 10-fold cross-validation. That is, estimate models with $\alpha = 0, 0.1, 0.2, \ldots, 1$ using the same values for λ across each model. Select the combination of α and λ with the lowest cross-validation MSE. For that combination, report the test MSE along with the number of non-zero coefficient estimates.

In [172]:

```
# elastic net regression
# L1 ratio is alpha in sklearn
ela = ElasticNetCV(l1_ratio = np.linspace(0.1, 1, 10), max_iter=iteratio
ns) #set range

ela_alpha, ela_error_train, ela_error_test, ela_model = get_lambda_MSE(e
la)

coefficients_elastic = sum(np.abs(ela_model.coef_)<0.1)

print("###")
print("Elastic Net 1")
print(coefficients_elastic, 'coefficients are non-zero estimates')
print("###")
```

```
/Users/reginacatipon/anaconda3/lib/python3.7/site-packages/sklearn/mode
l_selection/_split.py:1978: FutureWarning: The default value of cv will
change from 3 to 5 in version 0.22. Specify it explicitly to silence th
is warning.
  warnings.warn(CV_WARNING, FutureWarning)

###
Elastic Net 1
60 coefficients are non-zero estimates
###
```

In [128]:

```
#second elastic model
ela2 = ElasticNetCV(alphas=[.1, *list(np.linspace(0.12, 0.45, 5)),
                            .50, .75, .90, .95, .99, 1],
                    l1_ratio = [elamod.l1_ratio_],
                    max_iter=iters)

ela_alpha2, ela_error_train2, ela_error_test2, ela_model2 = get_lambda_M
SE(ela2)

coefficients_elastic2 = sum(np.abs(ela_model2.coef_)<0.1)

print("###")
print("ElasticNet 2")
print(coefficients_elastic2, 'coefficients are non-zero estimates')
print("###")
```

```
###
ElasticNet 2
60 coefficients are in range of -0.1 to 0.1
###

/Users/reginacatipon/anaconda3/lib/python3.7/site-packages/sklearn/mode
l_selection/_split.py:1978: FutureWarning: The default value of cv will
change from 3 to 5 in version 0.22. Specify it explicitly to silence th
is warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

In [167]:

```
rdg_alpha, las_alpha, ela_alpha, ela_alpha2 = [round(i, 2) for i in [rdg
_alpha, las_alpha, ela_alpha, ela_alpha2]]
```

In [173]:
```
print('Linear Regression: Train Error -', ols_error_train,
      ', Test Error -',ols_error_train)
print("Ridge Regression alpha =", rdg_alpha,
      ': Train Error -', rdg_error_train,
      ', Test Error -',rdg_error_test)
print('Lasso Regression alpha =', las_alpha,
      ': Train Error -', las_error_train,
      ', Test Error -',las_error_test )
print('ElasticNet alpha =', ela_alpha,
      ': Train Error -', ela_error_train,
      ', Test Error -',ela_error_test
      , ", L1 ratio = ", ela_model.l1_ratio_)
print('ElasticNet alpha =', ela_alpha2,
      ': Train Error -', ela_error_train2,
      ', Test Error -',ela_error_test2,
      ", L1 ratio = ", ela_model2.l1_ratio_)
```

```
Linear Regression: Train Error - 55.12263854924573 , Test Error - 55.12
263854924573
Ridge Regression alpha = 2.0 : Train Error - 55.14420678407385 , Test E
rror - 62.931657377495824
Lasso Regression alpha = 0.11 : Train Error - 57.39124881542893 , Test
Error - 62.86402674271178
ElasticNet alpha = 0.12318488951286882 : Train Error - 57.5029215884534
96 , Test Error - 62.92344587895507 , L1 ratio =  1.0
ElasticNet alpha = 0.12 : Train Error - 57.45916216451363 , Test Error
- 62.89960548404518 , L1 ratio =  1.0
```

In [174]:
```
print("###")
print(Y_test.shape[0], 'n sample size')
print("Mean egalit_scale", Y_test.mean())
print( "SD", Y_test.std())
print("###")
```

```
###
493 n sample size
Mean egalit_scale 19.113590263691684
SD 9.51567373529867
###
```

5. (5 points) Comment on the results obtained. How accurately can we predict an individual's egalitarianism? Is there much difference among the test errors resulting from these approaches?

From the results obtains, it looks like we can do a pretty good job in accurately predicting an individual's egalitarianism thanks to the marginal improvements from regularization. Out of the approaches, the ElasticNet gave us the best results. Overall, with an increase in regularization, training MSE will increase but testing MSE will decrease.

In [ ]: