Homework 4

April 10, 2019

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings("ignore")
```

```
In [2]: names = ['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
                'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
                'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
                'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
                'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
                'fractal_dimension_se', 'radius_worst', 'texture_worst',
                'perimeter_worst', 'area_worst', 'smoothness_worst',
                'compactness_worst', 'concavity_worst', 'concave points_worst',
                'symmetry_worst', 'fractal_dimension_worst']
```

```
In [3]: df = pd.read_csv("wdbc.csv", names = names)
```

```
In [4]: df.head()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
0	0.11840	0.27760	0.3001	0.14710	
1	0.08474	0.07864	0.0869	0.07017	
2	0.10960	0.15990	0.1974	0.12790	
3	0.14250	0.28390	0.2414	0.10520	
4	0.10030	0.13280	0.1980	0.10430	

	...	radius_worst	texture_worst	perimeter_worst	\
0	...	25.38	17.33	184.60	
1	...	24.99	23.41	158.80	

```
2           ...                    23.57          25.53           152.50
3           ...                    14.91          26.50            98.87
4           ...                    22.54          16.67           152.20

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 32 columns]
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
```

```
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

```
In [6]: df = df.loc[:,df.columns != 'id']
```

```
In [7]: df.head()
```

Out[7]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	M	17.99	10.38	122.80	1001.0	
1	M	20.57	17.77	132.90	1326.0	
2	M	19.69	21.25	130.00	1203.0	
3	M	11.42	20.38	77.58	386.1	
4	M	20.29	14.34	135.10	1297.0	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
0	0.11840	0.27760	0.3001	0.14710	
1	0.08474	0.07864	0.0869	0.07017	
2	0.10960	0.15990	0.1974	0.12790	
3	0.14250	0.28390	0.2414	0.10520	
4	0.10030	0.13280	0.1980	0.10430	

	symmetry_mean	...	radius_worst	texture_worst	\
0	0.2419	...	25.38	17.33	
1	0.1812	...	24.99	23.41	
2	0.2069	...	23.57	25.53	
3	0.2597	...	14.91	26.50	
4	0.1809	...	22.54	16.67	

	perimeter_worst	area_worst	smoothness_worst	compactness_worst	\
0	184.60	2019.0	0.1622	0.6656	
1	158.80	1956.0	0.1238	0.1866	
2	152.50	1709.0	0.1444	0.4245	
3	98.87	567.7	0.2098	0.8663	
4	152.20	1575.0	0.1374	0.2050	

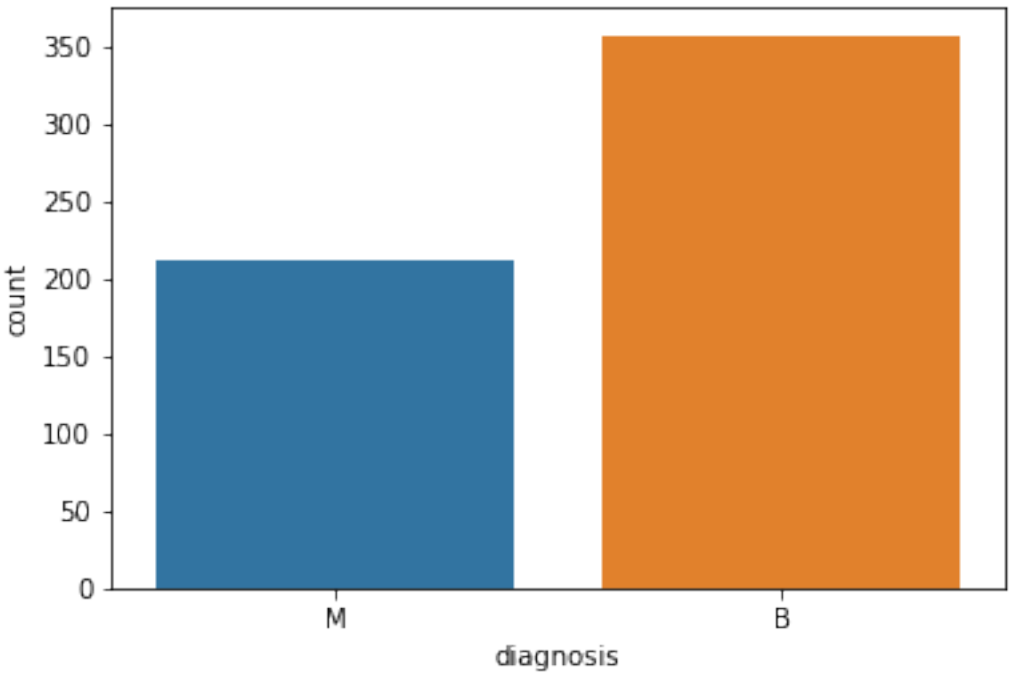
	concavity_worst	concave points_worst	symmetry_worst	\
0	0.7119	0.2654	0.4601	
1	0.2416	0.1860	0.2750	
2	0.4504	0.2430	0.3613	
3	0.6869	0.2575	0.6638	
4	0.4000	0.1625	0.2364	

	fractal_dimension_worst

```
0                    0.11890
1                    0.08902
2                    0.08758
3                    0.17300
4                    0.07678

[5 rows x 31 columns]
```

0.0.1 Exploratory Data Analysis

```
In [8]: sns.countplot(df.diagnosis, label = "Count")
        df.diagnosis.value_counts()

Out[8]: B    357
        M    212
        Name: diagnosis, dtype: int64
```




```
In [9]: df.describe()

Out[9]:        radius_mean  texture_mean  perimeter_mean    area_mean  \
        count   569.000000    569.000000      569.000000   569.000000
        mean     14.127292     19.289649       91.969033   654.889104
        std       3.524049      4.301036       24.298981   351.914129
        min       6.981000      9.710000       43.790000   143.500000
        25%      11.700000     16.170000       75.170000   420.300000
```

```
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
max      28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       symmetry_mean  fractal_dimension_mean           ...             \
count     569.000000              569.000000           ...
mean        0.181162                0.062798           ...
std         0.027414                0.007060           ...
min         0.106000                0.049960           ...
25%         0.161900                0.057700           ...
50%         0.179200                0.061540           ...
75%         0.195700                0.066120           ...
max         0.304000                0.097440           ...

       radius_worst  texture_worst  perimeter_worst   area_worst  \
count    569.000000     569.000000       569.000000   569.000000
mean      16.269190      25.677223       107.261213   880.583128
std        4.833242       6.146258        33.602542   569.356993
min        7.930000      12.020000        50.410000   185.200000
25%       13.010000      21.080000        84.110000   515.300000
50%       14.970000      25.410000        97.660000   686.500000
75%       18.790000      29.720000       125.400000  1084.000000
max       36.040000      49.540000       251.200000  4254.000000

       smoothness_worst  compactness_worst  concavity_worst  \
count        569.000000         569.000000       569.000000
mean           0.132369           0.254265         0.272188
std            0.022832           0.157336         0.208624
min            0.071170           0.027290         0.000000
25%            0.116600           0.147200         0.114500
50%            0.131300           0.211900         0.226700
75%            0.146000           0.339100         0.382900
max            0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst
count            569.000000      569.000000               569.000000
mean               0.114606        0.290076                 0.083946
std                0.065732        0.061867                 0.018061
```

```
        min             0.000000        0.156500                0.055040
        25%             0.064930        0.250400                0.071460
        50%             0.099930        0.282200                0.080040
        75%             0.161400        0.317900                0.092080
        max             0.291000        0.663800                0.207500

        [8 rows x 30 columns]
```

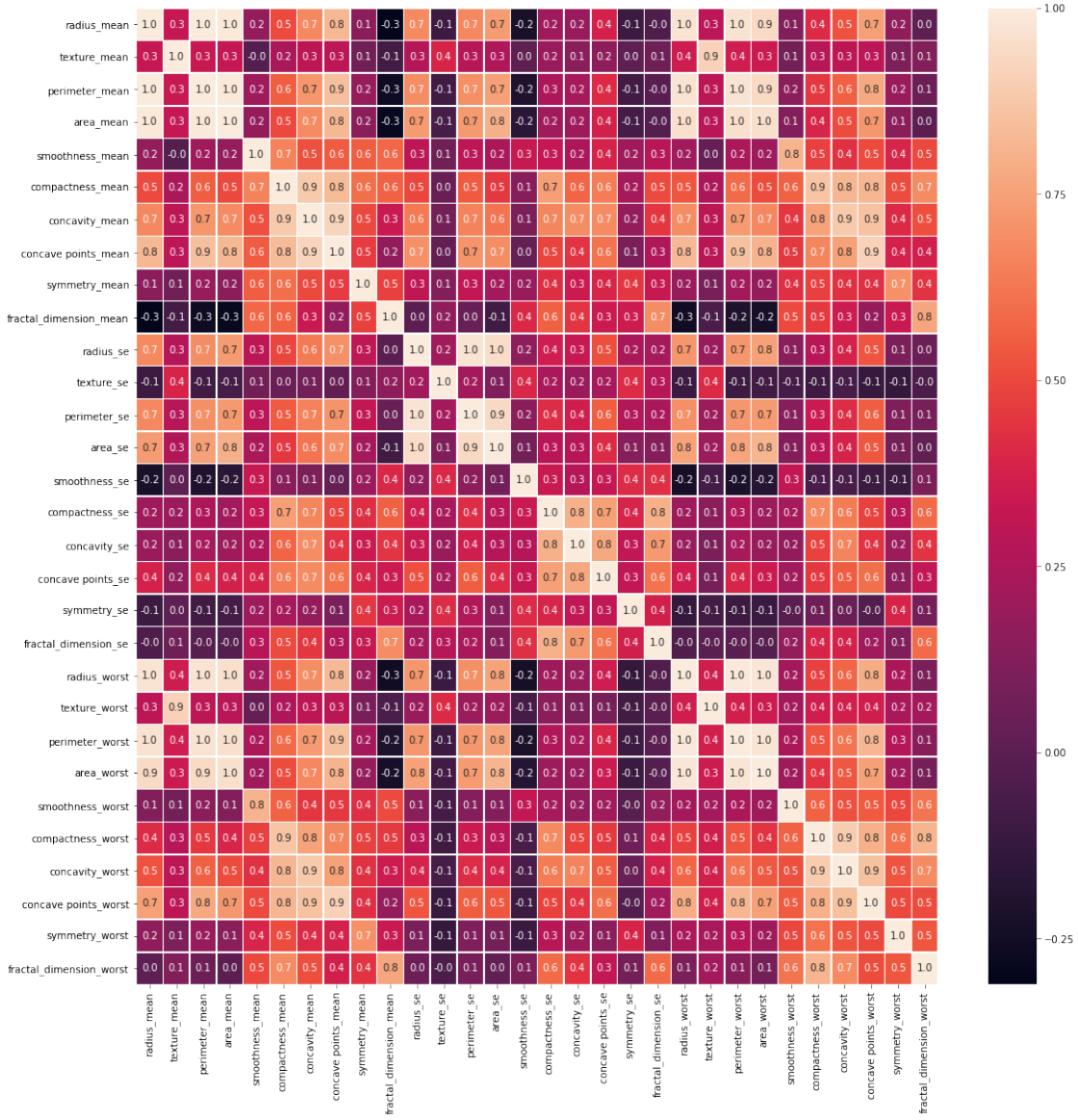
```
In [10]: from sklearn.preprocessing import LabelEncoder
```

```
In [11]: label_encoder = LabelEncoder()
         df.iloc[:,0] = label_encoder.fit_transform(df.iloc[:,0]).astype('float64')
```

```
In [12]: cor = df.loc[:, df.columns != "diagnosis"].corr()
```

```
In [13]: f,ax = plt.subplots(figsize = (18,18))
         sns.heatmap(cor, annot = True, linewidths = .5, fmt='.1f', ax = ax)
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x10f33ff28>
```



All Mean Columns

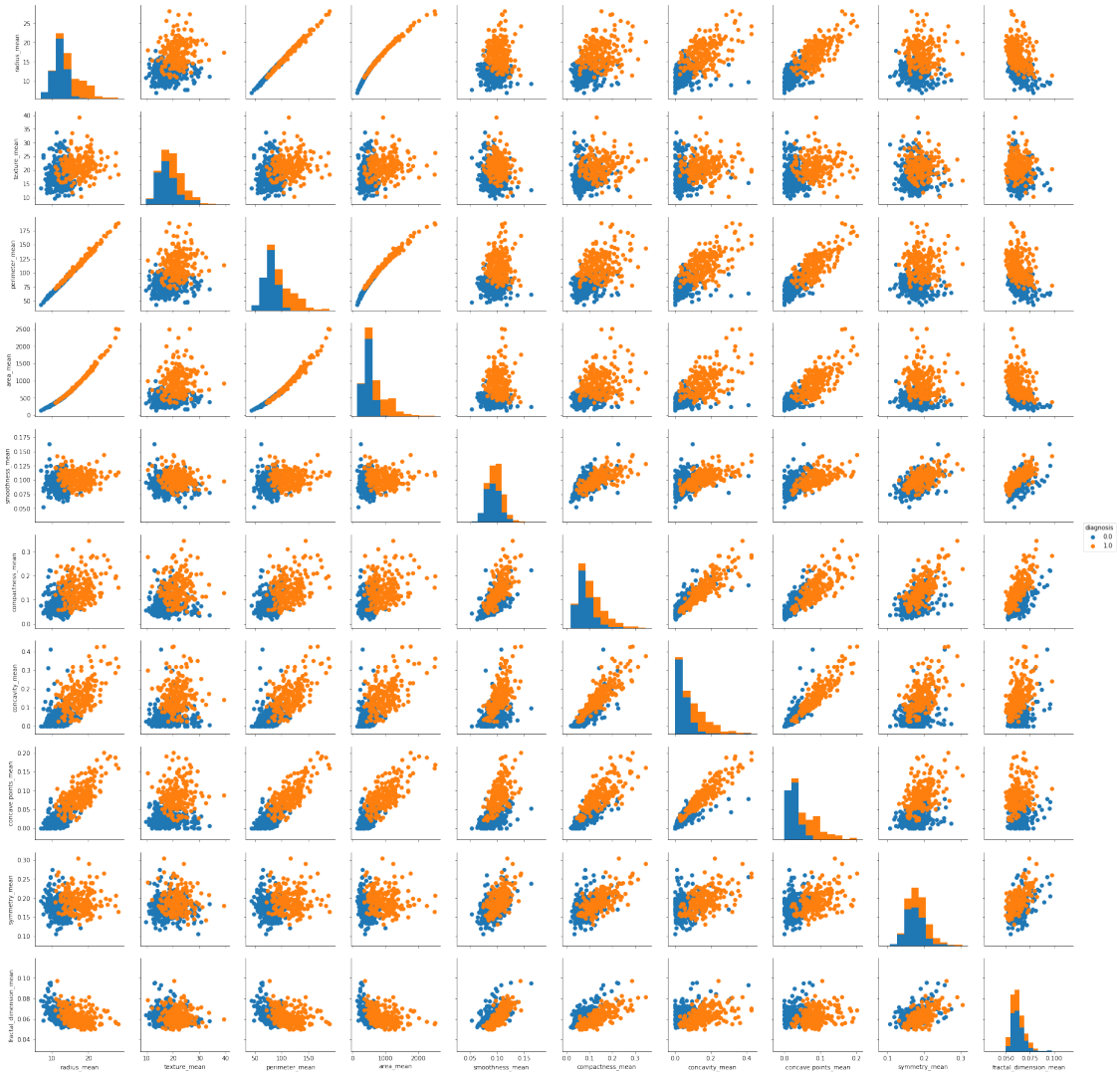
```
In [14]: features_mean= df.iloc[:,0:11]
         features_mean.head()
```

```
Out[14]:    diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
         0        1.0        17.99         10.38          122.80     1001.0
         1        1.0        20.57         17.77          132.90     1326.0
         2        1.0        19.69         21.25          130.00     1203.0
         3        1.0        11.42         20.38           77.58      386.1
         4        1.0        20.29         14.34          135.10     1297.0
```

```
   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   symmetry_mean  fractal_dimension_mean
0         0.2419                 0.07871
1         0.1812                 0.05667
2         0.2069                 0.05999
3         0.2597                 0.09744
4         0.1809                 0.05883
```

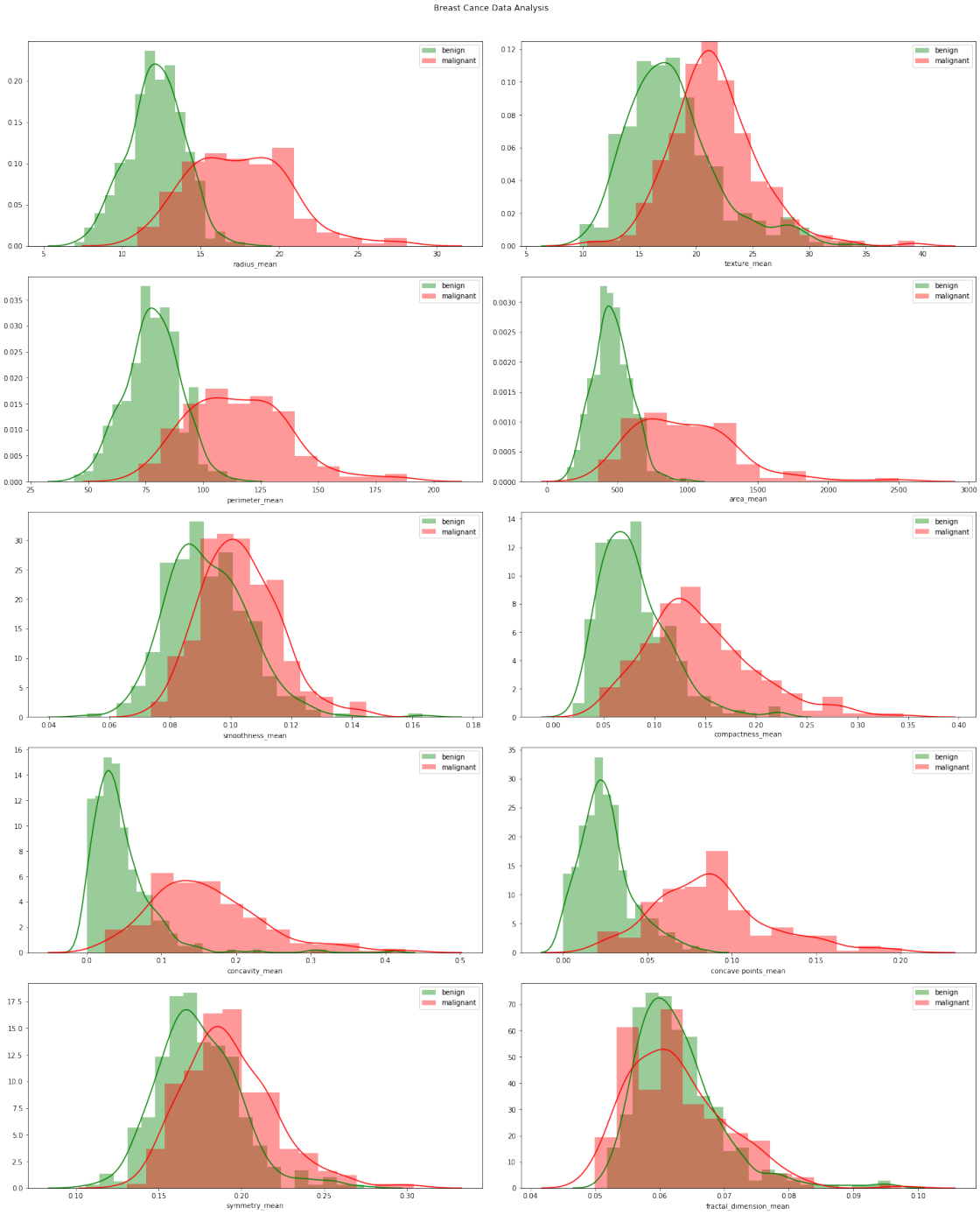
In [15]:
```
g = sns.PairGrid(features_mean, hue ="diagnosis", vars = ['radius_mean', 'texture_mean',
    'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
    'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean'])
g = g.map_offdiag(plt.scatter)
g = g.map_diag(plt.hist)
g = g.add_legend()
g
```

Out[15]:
```
<seaborn.axisgrid.PairGrid at 0x1a21dbfe48>
```




```
In [16]: res = pd.DataFrame()
         res['diagnosis'] = features_mean.iloc[:,0]
         fm = features_mean.loc[:, features_mean.columns != "diagnosis"]

In [17]: fig = plt.figure(figsize = (20, 25))
         j = 0
         for i in fm.columns:
             plt.subplot(5, 2, j+1)
             j += 1
             sns.distplot(fm[i][res['diagnosis']==0], color='g', label = 'benign')
             sns.distplot(fm[i][res['diagnosis']==1], color='r', label = 'malignant')
             plt.legend(loc = "best")
         fig.suptitle('Breast Cance Data Analysis')
         fig.tight_layout()
```

```
fig.subplots_adjust(top=0.95)
plt.show()
```



All se Columns

In [18]:
```
features_se = df.iloc[:, np.r_[0, 11:21]]
features_se.head()
```

Out[18]:

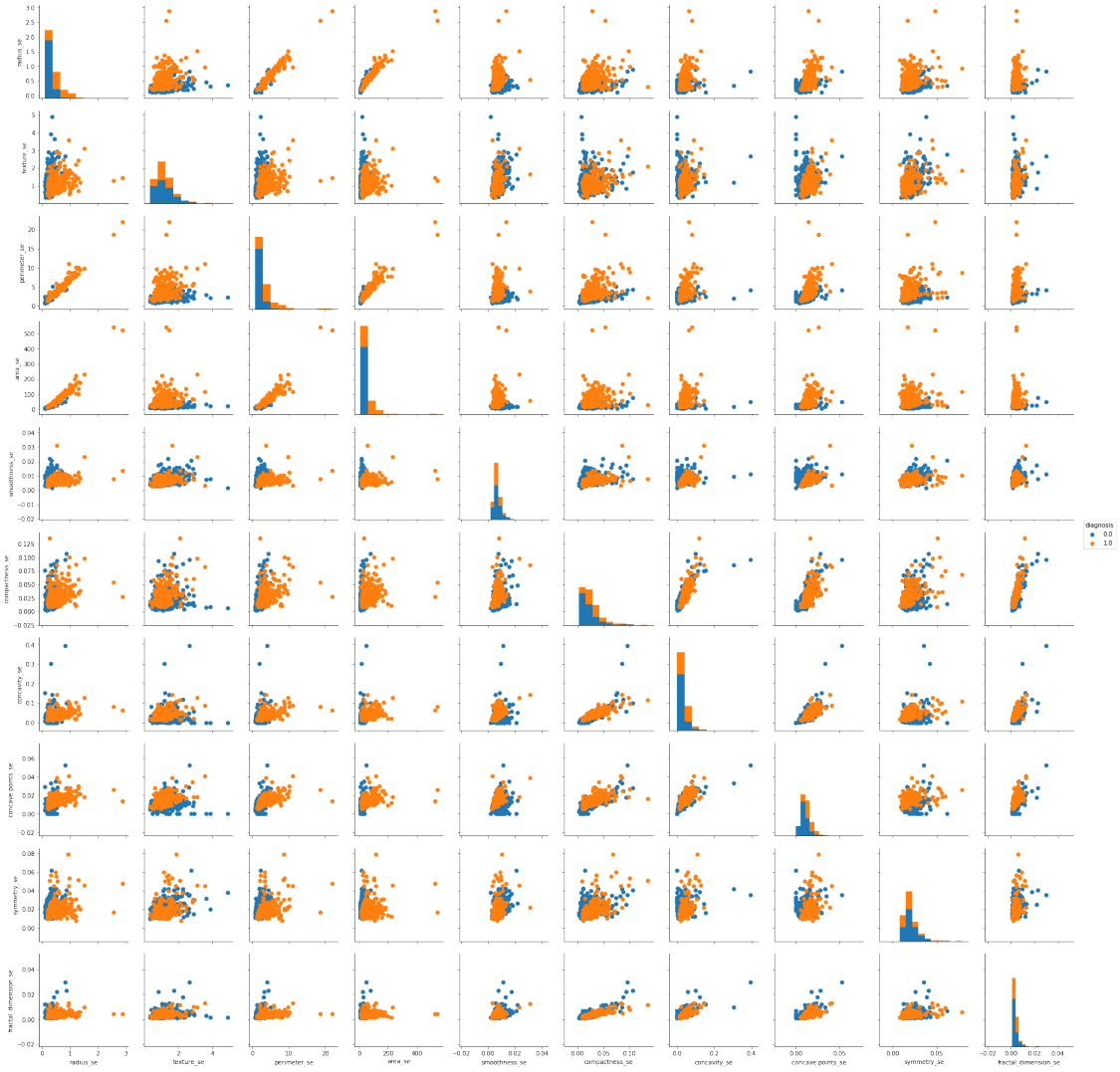
	diagnosis	radius_se	texture_se	perimeter_se	area_se	smoothness_se	\
0	1.0	1.0950	0.9053	8.589	153.40	0.006399	
1	1.0	0.5435	0.7339	3.398	74.08	0.005225	
2	1.0	0.7456	0.7869	4.585	94.03	0.006150	
3	1.0	0.4956	1.1560	3.445	27.23	0.009110	
4	1.0	0.7572	0.7813	5.438	94.44	0.011490	

	compactness_se	concavity_se	concave points_se	symmetry_se	\
0	0.04904	0.05373	0.01587	0.03003	
1	0.01308	0.01860	0.01340	0.01389	
2	0.04006	0.03832	0.02058	0.02250	
3	0.07458	0.05661	0.01867	0.05963	
4	0.02461	0.05688	0.01885	0.01756	

	fractal_dimension_se
0	0.006193
1	0.003532
2	0.004571
3	0.009208
4	0.005115

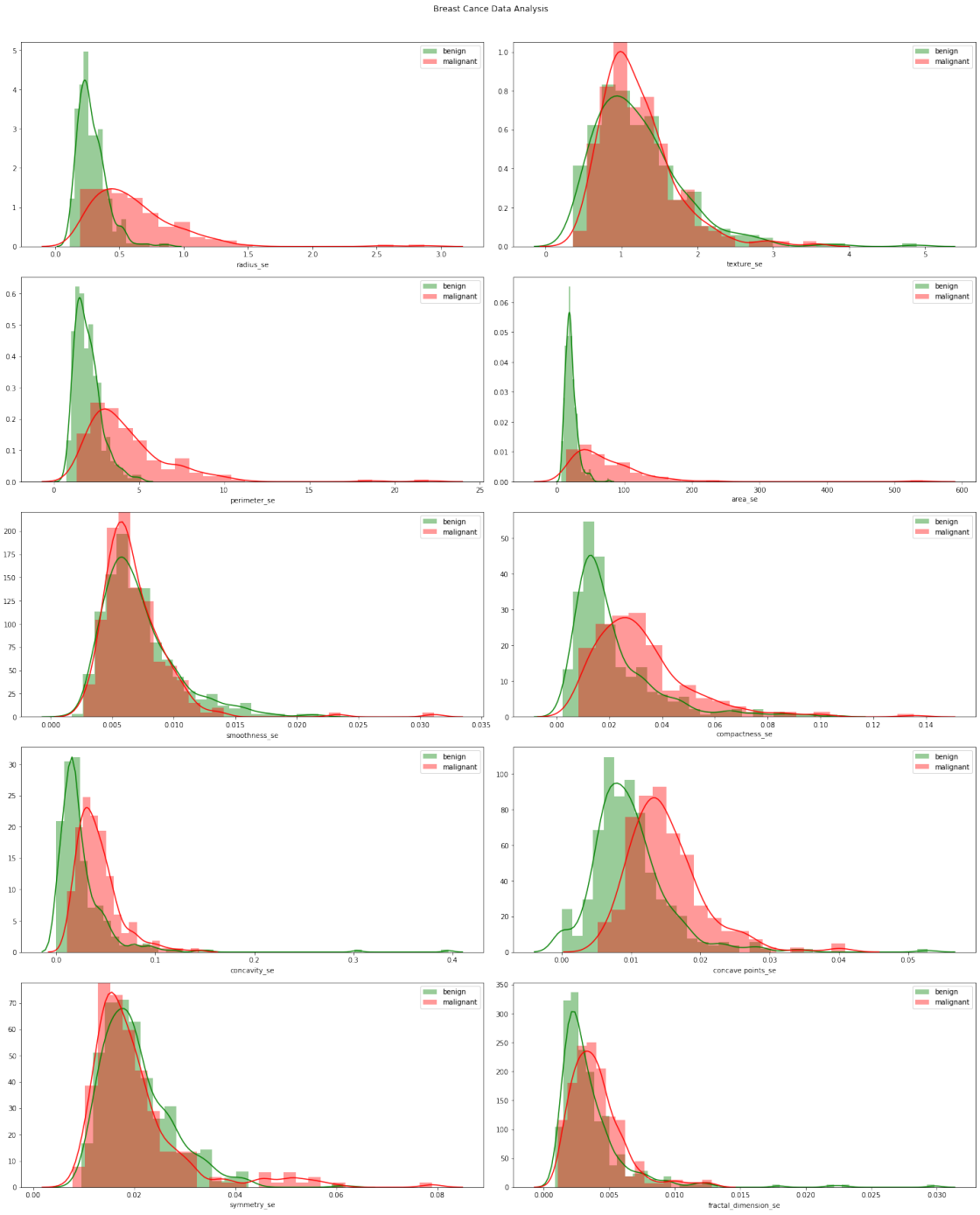
In [19]:
```
g = sns.PairGrid(features_se, hue ="diagnosis", vars = ['radius_se', 'texture_se', 'pe
    'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se',
    'symmetry_se', 'fractal_dimension_se'])
g = g.map_offdiag(plt.scatter)
g = g.map_diag(plt.hist)
g = g.add_legend()
g
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x1a24a20668>




```
In [20]: res_fse = pd.DataFrame()
         res_fse['diagnosis'] = features_se.iloc[:,0]
         fse = features_se.loc[:, features_se.columns != "diagnosis"]

         fig = plt.figure(figsize = (20, 25))
         j = 0
         for i in fse.columns:
             plt.subplot(5, 2, j+1)
             j += 1
             sns.distplot(fse[i][res_fse['diagnosis']==0], color='g', label = 'benign')
             sns.distplot(fse[i][res_fse['diagnosis']==1], color='r', label = 'malignant')
             plt.legend(loc = "best")
         fig.suptitle('Breast Cance Data Analysis')
         fig.tight_layout()
```

```
fig.subplots_adjust(top=0.95)
plt.show()
```



All Worst Columns

```
In [21]: features_worst = df.iloc[:, np.r_[0, 21:31]]
         features_worst.head()
```

Out[21]:

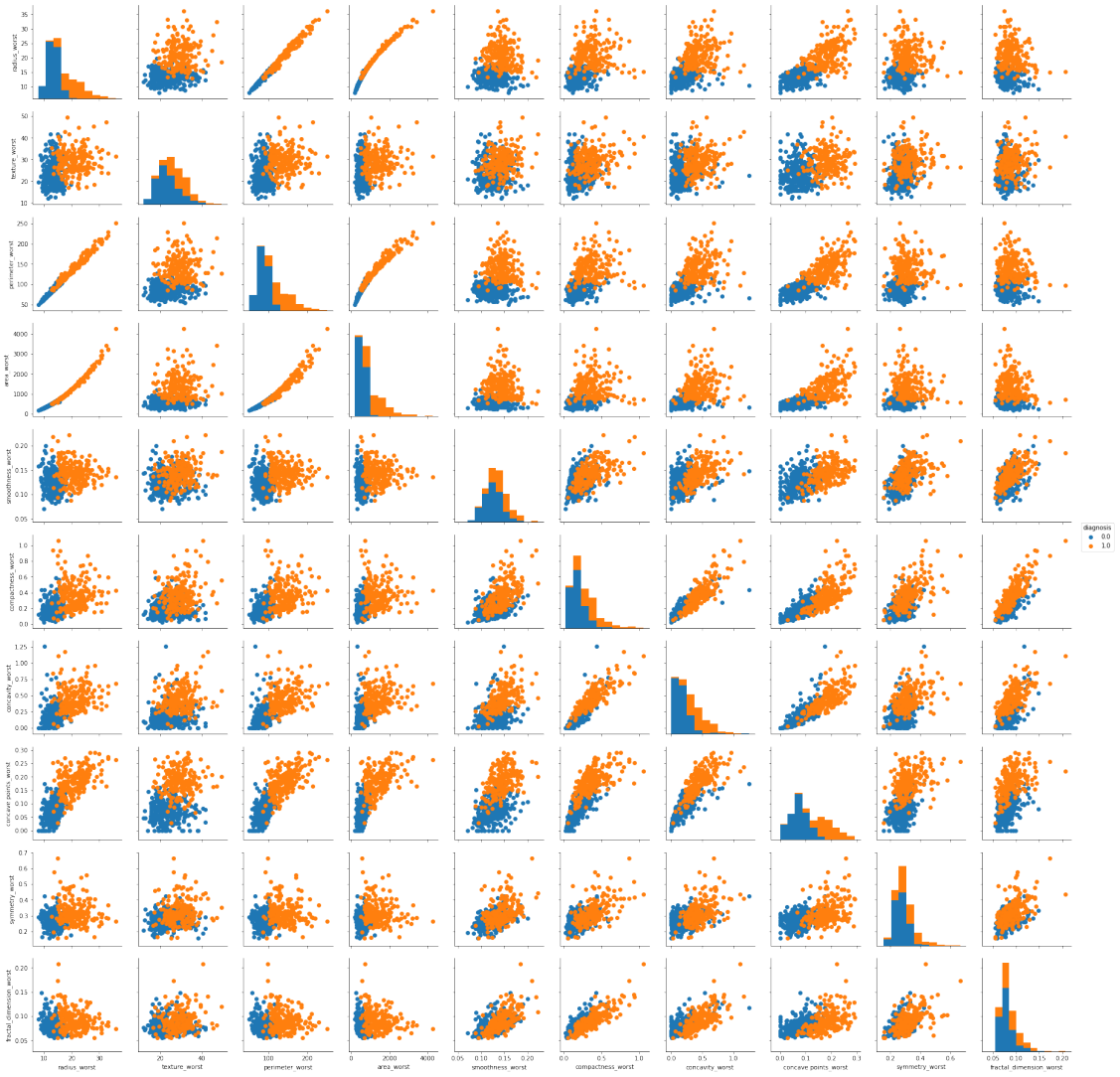
	diagnosis	radius_worst	texture_worst	perimeter_worst	area_worst	\
0	1.0	25.38	17.33	184.60	2019.0	
1	1.0	24.99	23.41	158.80	1956.0	
2	1.0	23.57	25.53	152.50	1709.0	
3	1.0	14.91	26.50	98.87	567.7	
4	1.0	22.54	16.67	152.20	1575.0	

	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	\
0	0.1622	0.6656	0.7119	0.2654	
1	0.1238	0.1866	0.2416	0.1860	
2	0.1444	0.4245	0.4504	0.2430	
3	0.2098	0.8663	0.6869	0.2575	
4	0.1374	0.2050	0.4000	0.1625	

	symmetry_worst	fractal_dimension_worst
0	0.4601	0.11890
1	0.2750	0.08902
2	0.3613	0.08758
3	0.6638	0.17300
4	0.2364	0.07678

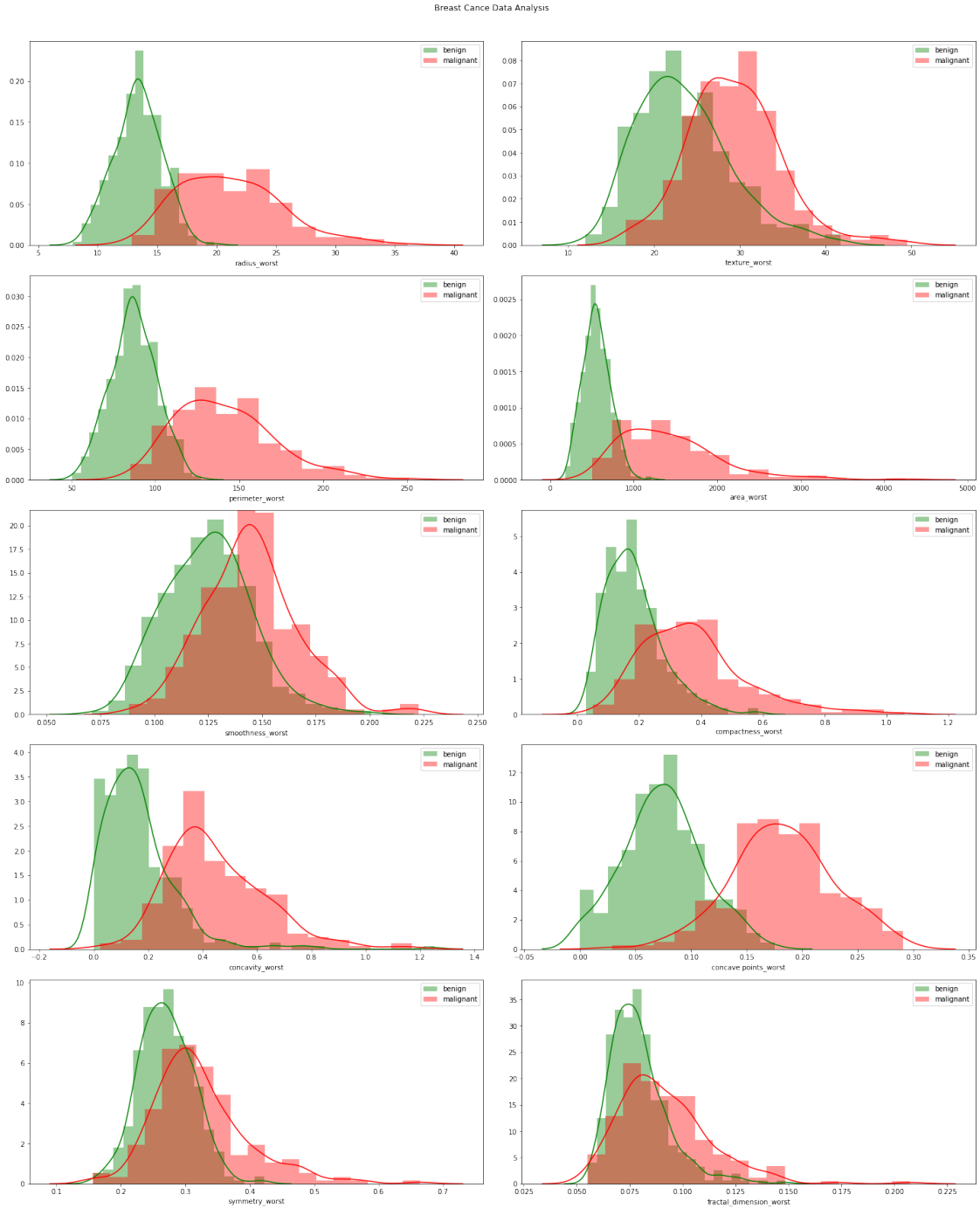
In [22]:
```
g = sns.PairGrid(features_worst, hue ="diagnosis", vars = ['radius_worst', 'texture_w
    'area_worst', 'smoothness_worst', 'compactness_worst',
    'concavity_worst', 'concave points_worst', 'symmetry_worst',
    'fractal_dimension_worst'])
g = g.map_offdiag(plt.scatter)
g = g.map_diag(plt.hist)
g = g.add_legend()
g
```

Out[22]: <seaborn.axisgrid.PairGrid at 0x1a2c182f28>




```
In [23]: res_w = pd.DataFrame()
         res_w['diagnosis'] = features_worst.iloc[:,0]
         fw = features_worst.loc[:, features_worst.columns != "diagnosis"]

         fig = plt.figure(figsize = (20, 25))
         j = 0
         for i in fw.columns:
             plt.subplot(5, 2, j+1)
             j += 1
             sns.distplot(fw[i][res_w['diagnosis']==0], color='g', label = 'benign')
             sns.distplot(fw[i][res_w['diagnosis']==1], color='r', label = 'malignant')
             plt.legend(loc = "best")
         fig.suptitle('Breast Cance Data Analysis')
         fig.tight_layout()
```

```
fig.subplots_adjust(top=0.95)
plt.show()
```



0.0.2 Remove one of two features that have a correlation higher than 0.9

```
In [24]: cor2 = df.corr()
         columns = np.full((cor2.shape[0],), True, dtype=bool)
         for i in range(cor2.shape[0]):
             for j in range(i+1, cor2.shape[0]):
                 if cor2.iloc[i,j] >= 0.9:
                     if columns[j]:
                         columns[j] = False
```

```
In [25]: col = df.columns[columns]
         df_cor = df[col]
         print(df_cor.shape)
         df_cor.head()
```

```
(569, 21)
```

Out[25]:

	diagnosis	radius_mean	texture_mean	smoothness_mean	compactness_mean	\
0	1.0	17.99	10.38	0.11840	0.27760	
1	1.0	20.57	17.77	0.08474	0.07864	
2	1.0	19.69	21.25	0.10960	0.15990	
3	1.0	11.42	20.38	0.14250	0.28390	
4	1.0	20.29	14.34	0.10030	0.13280	

	concavity_mean	symmetry_mean	fractal_dimension_mean	radius_se	\
0	0.3001	0.2419	0.07871	1.0950	
1	0.0869	0.1812	0.05667	0.5435	
2	0.1974	0.2069	0.05999	0.7456	
3	0.2414	0.2597	0.09744	0.4956	
4	0.1980	0.1809	0.05883	0.7572	

	texture_se	...	compactness_se	concavity_se	\
0	0.9053	...	0.04904	0.05373	
1	0.7339	...	0.01308	0.01860	
2	0.7869	...	0.04006	0.03832	
3	1.1560	...	0.07458	0.05661	
4	0.7813	...	0.02461	0.05688	

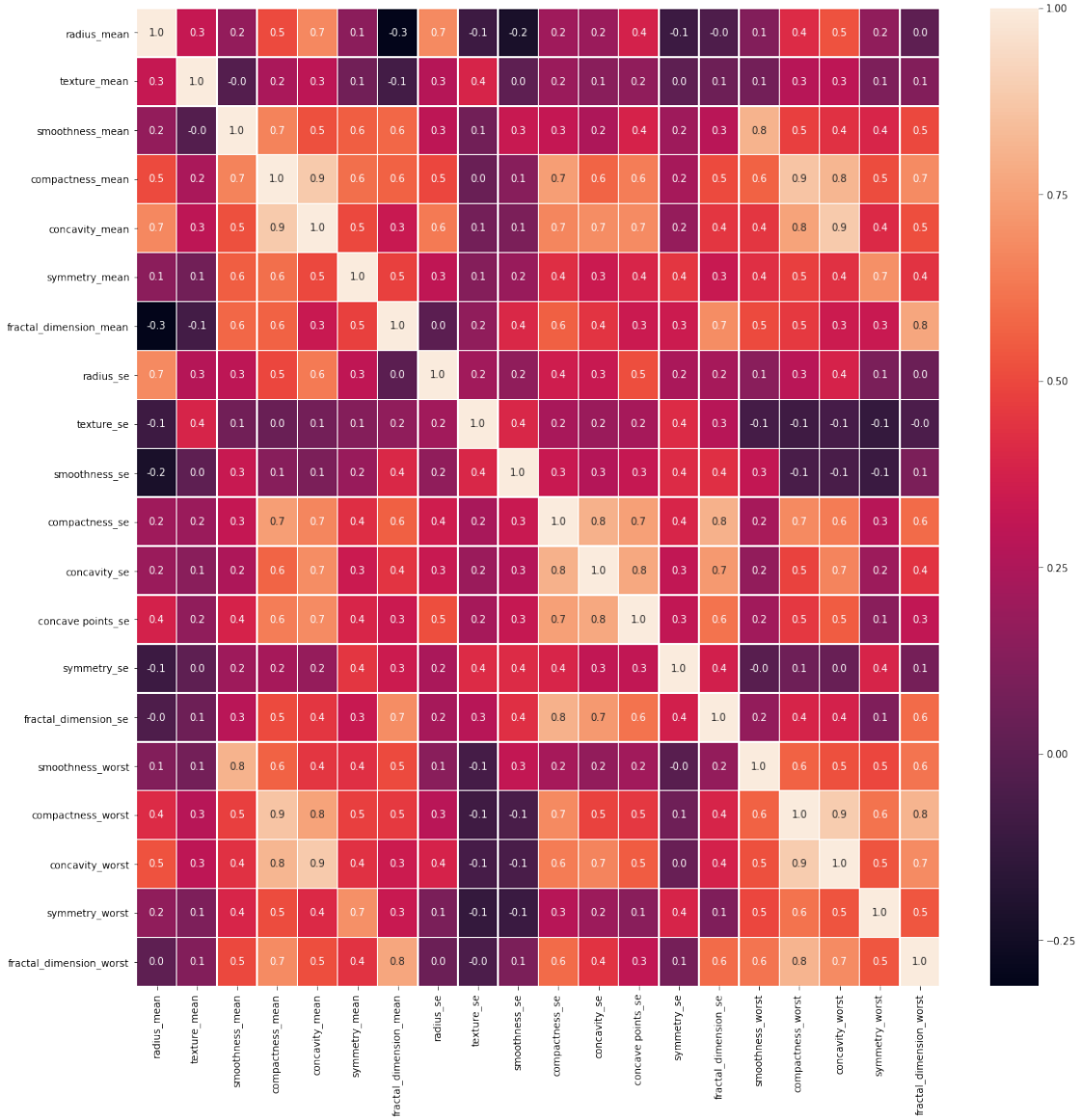
	concave points_se	symmetry_se	fractal_dimension_se	smoothness_worst	\
0	0.01587	0.03003	0.006193	0.1622	
1	0.01340	0.01389	0.003532	0.1238	
2	0.02058	0.02250	0.004571	0.1444	
3	0.01867	0.05963	0.009208	0.2098	
4	0.01885	0.01756	0.005115	0.1374	

	compactness_worst	concavity_worst	symmetry_worst	fractal_dimension_worst
0	0.6656	0.7119	0.4601	0.11890
1	0.1866	0.2416	0.2750	0.08902

```
2                0.4245                0.4504                0.3613                0.08758
3                0.8663                0.6869                0.6638                0.17300
4                0.2050                0.4000                0.2364                0.07678

[5 rows x 21 columns]
```

In [26]:
```
cor3 = df_cor.loc[:, df_cor.columns != "diagnosis"].corr()
f,ax = plt.subplots(figsize = (18,18))
sns.heatmap(cor3, annot = True, linewidths = .5, fmt='.1f', ax = ax)
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1a312f1978>



In [27]:
```
cor_features = df_cor.loc[:, df_cor.columns != 'diagnosis'].columns
```

```
In [28]: cor_features

Out[28]: Index(['radius_mean', 'texture_mean', 'smoothness_mean', 'compactness_mean',
                'concavity_mean', 'symmetry_mean', 'fractal_dimension_mean',
                'radius_se', 'texture_se', 'smoothness_se', 'compactness_se',
                'concavity_se', 'concave points_se', 'symmetry_se',
                'fractal_dimension_se', 'smoothness_worst', 'compactness_worst',
                'concavity_worst', 'symmetry_worst', 'fractal_dimension_worst'],
               dtype='object')
```

Split Data

```
In [29]: ### ALL Columns
         X = df.loc[:, df.columns != "diagnosis"]
         y = df.loc[:, df.columns == "diagnosis"]

In [30]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state

In [31]: X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.20, ra

In [32]: X_val.shape, X_test.shape, X_train.shape

Out[32]: ((91, 30), (114, 30), (364, 30))

In [33]: from sklearn.preprocessing import MinMaxScaler

In [34]: scaler = MinMaxScaler()
         scaler.fit(X_train)

Out[34]: MinMaxScaler(copy=True, feature_range=(0, 1))

In [35]: X_train = scaler.transform(X_train)
         X_val = scaler.transform(X_val)
         X_test = scaler.transform(X_test)
```

0.0.3 Feature Selection

```
In [36]: from sklearn.ensemble import RandomForestClassifier

In [37]: feat = RandomForestClassifier(random_state=10)
         feat.fit(X_train, y_train)

Out[37]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                     oob_score=False, random_state=10, verbose=0, warm_start=False)
```

```
In [38]: feat_df = pd.DataFrame(feat.feature_importances_, index = X.columns, columns = ["Impo
```

```
In [39]: top_10 = feat_df.sort_values("Importance", ascending=False).head(10)
         top_10
```

Out[39]:

	Importance
concave points_worst	0.237546
area_worst	0.128653
area_mean	0.122280
concave points_mean	0.084362
radius_mean	0.081335
concavity_mean	0.071055
perimeter_se	0.048826
texture_worst	0.023309
radius_worst	0.022058
fractal_dimension_worst	0.020693

0.1 4 Models

- Model 1 = All features
- Model 2 = All features without the high correlated features
- Model 3 = Random Forest Feature Importance
- Model 4 = Features Selected based on EDA

```
In [40]: names = X.columns
         X_train = pd.DataFrame(X_train, columns=names)
         X_train.head()
```

Out[40]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	\
0	0.486961	0.651867	0.474121	0.333107	0.333225	
1	0.318472	0.461411	0.320710	0.184263	0.719778	
2	0.578778	0.367635	0.564647	0.427784	0.572065	
3	0.253632	0.106224	0.242900	0.137858	0.406213	
4	0.402717	0.425726	0.405017	0.255016	0.688281	

	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	\
0	0.342361	0.282099	0.285089	0.205556	
1	0.542870	0.219447	0.297465	0.573737	
2	0.401466	0.309981	0.447018	0.432828	
3	0.163437	0.069306	0.131561	0.317677	
4	0.511821	0.443065	0.452932	0.438889	

	fractal_dimension_mean	...	radius_worst	\
0	0.086563	...	0.442903	
1	0.517060	...	0.324795	
2	0.203243	...	0.647812	
3	0.171019	...	0.202063	
4	0.368155	...	0.472074	

```
   texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0       0.624733         0.410329    0.270055          0.478307
1       0.429638         0.299766    0.174941          0.622268
2       0.429638         0.596095    0.481665          0.602457
3       0.193230         0.183326    0.093320          0.383213
4       0.463486         0.456646    0.288488          0.640098

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0           0.373442         0.402236              0.599035        0.184309
1           0.330753         0.213898              0.535997        0.321506
2           0.314162         0.309824              0.720289        0.388725
3           0.174744         0.143051              0.368584        0.304554
4           0.353164         0.443530              0.730623        0.319732

   fractal_dimension_worst
0                 0.160042
1                 0.393939
2                 0.182999
3                 0.136954
4                 0.307359

[5 rows x 30 columns]
```

In [41]:
```
names = X.columns
X_val = pd.DataFrame(X_val, columns=names)
X_val.head()
```

Out[41]:
```
   radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
0     0.344030      0.516183        0.363693   0.213022         0.567720
1     0.536182      0.368050        0.516965   0.380700         0.361138
2     0.127124      0.364315        0.122314   0.061760         0.400022
3     0.321785      0.252282        0.308064   0.187656         0.386011
4     0.433007      0.339004        0.436805   0.281527         0.557945

   compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
0          0.678662        0.500234             0.430070       0.448990
1          0.244277        0.191401             0.288966       0.283333
2          0.161604        0.069072             0.075249       0.594949
3          0.178812        0.024719             0.049389       0.174242
4          0.510699        0.317245             0.385288       0.473737

   fractal_dimension_mean          ...            radius_worst  \
0                0.483572          ...                0.346496
1                0.090354          ...                0.475987
2                0.298441          ...                0.114194
3                0.179444          ...                0.261117
4                0.319924          ...                0.436855
```

```
   texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0       0.466151         0.342099    0.190302          0.613023
1       0.382196         0.442203    0.301022          0.344912
2       0.362473         0.101947    0.049155          0.344912
3       0.146055         0.236516    0.128146          0.237932
4       0.406183         0.409831    0.264402          0.484911

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0           0.579610         0.552875              0.614881        0.252119
1           0.210738         0.282588              0.541164        0.323477
2           0.123129         0.101997              0.225973        0.317169
3           0.077432         0.028091              0.114089        0.057954
4           0.536630         0.421246              0.642094        0.533215

   fractal_dimension_worst
0                 0.381477
1                 0.094057
2                 0.198085
3                 0.085662
4                 0.447724

[5 rows x 30 columns]
```

In [42]:
```
names = X.columns
X_test = pd.DataFrame(X_test, columns=names)
X_test.head()
```

Out[42]:
```
   radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
0     0.303800      0.448548        0.309930   0.175270         0.629630
1     0.294808      0.644813        0.278557   0.167296         0.383187
2     0.333144      0.246888        0.316495   0.196394         0.293581
3     0.344503      0.351037        0.327759   0.207678         0.142609
4     0.286289      0.361411        0.268261   0.161315         0.404040

   compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
0          0.477031        0.338566             0.406163       0.533333
1          0.122213        0.064948             0.102783       0.282323
2          0.136279        0.048899             0.131809       0.267172
3          0.122774        0.057990             0.068290       0.290404
4          0.068382        0.060028             0.145278       0.205556

   fractal_dimension_mean          ...            radius_worst  \
0                0.490522          ...                0.301672
1                0.123842          ...                0.228388
2                0.124263          ...                0.248310
3                0.124263          ...                0.294913
4                0.182603          ...                0.191035
```

```
   texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0       0.470149         0.313213    0.162013          0.569438
1       0.591951         0.203596    0.110032          0.381232
2       0.194296         0.229693    0.123796          0.212838
3       0.352878         0.275860    0.155943          0.153734
4       0.287580         0.169580    0.088650          0.170640

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0           0.347634         0.407827              0.706511        0.398186
1           0.076656         0.111022              0.206855        0.173270
2           0.072193         0.050000              0.283018        0.112557
3           0.183767         0.125000              0.259387        0.211118
4           0.018337         0.038602              0.172683        0.083185

   fractal_dimension_worst
0                 0.366391
1                 0.084219
2                 0.079103
3                 0.142464
4                 0.043618

[5 rows x 30 columns]
```

```
In [43]: rf_features = top_10.T.columns
         rf_features
```

```
Out[43]: Index(['concave points_worst', 'area_worst', 'area_mean',
                'concave points_mean', 'radius_mean', 'concavity_mean', 'perimeter_se',
                'texture_worst', 'radius_worst', 'fractal_dimension_worst'],
               dtype='object')
```

```
In [44]: eda_features = ['radius_mean', 'perimeter_mean',  'area_mean', 'concavity_mean', 'conc
                        'radius_worst', 'perimeter_worst','area_worst', 'concavity_worst', 'co
```

1 KNN

```
In [45]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import classification_report, confusion_matrix , accuracy_score
```

Model 1

```
In [46]: from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import roc_curve
         from sklearn.metrics import auc
```

```
In [47]: def knn_grid(x,y):
             knn2 = KNeighborsClassifier()
             parm_grid = {'n_neighbors':np.arange(1,25)}
```

```
            knn_grid = GridSearchCV(knn2, parm_grid, cv = 5)
            knn_grid.fit(x, y)
            return knn_grid.best_params_ , knn_grid.best_score_
```

```
In [48]: knn_grid(X_train, y_train)
```

```
Out[48]: ({'n_neighbors': 3}, 0.9532967032967034)
```

```
In [49]: knn_all = KNeighborsClassifier(n_neighbors=3)
```

```
In [50]: knn_all.fit(X_train, y_train)
```

```
Out[50]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                    weights='uniform')
```

```
In [51]: y_pred_all = knn_all.predict(X_val)
```

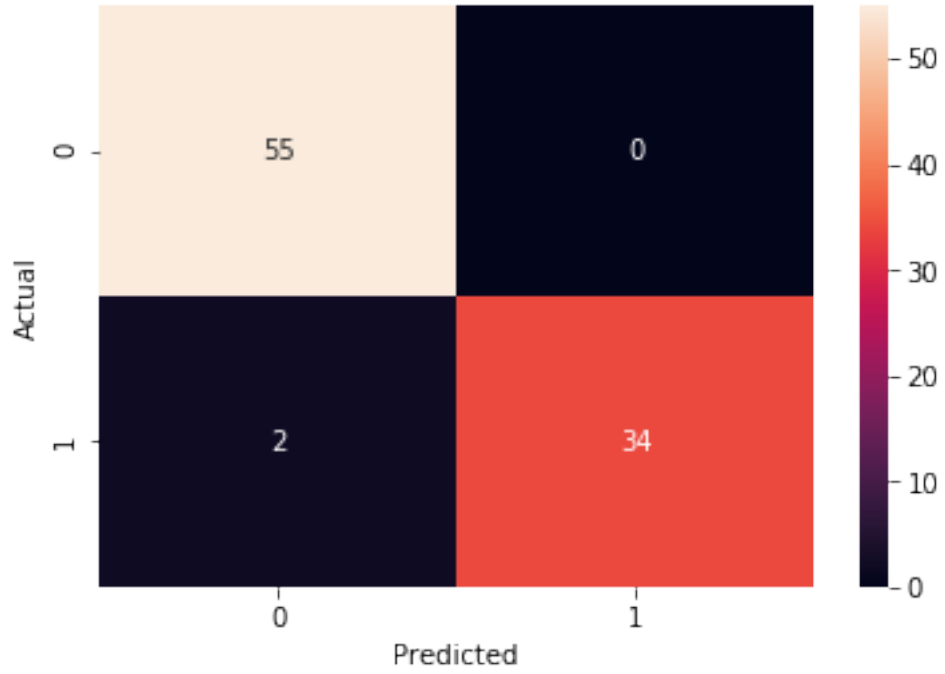
```
In [52]: def plot_roc_curve(fpr, tpr,y):
             plt.figure(figsize=(8,8))
             plt.title(y)
             plt.plot(fpr, tpr, linewidth=2)
             plt.plot([0, 1], [0, 1], 'k--')
             plt.axis([-0.005, 1, 0, 1.005])
             plt.xticks(np.arange(0,1, 0.05), rotation=90)
             plt.xlabel("False Positive Rate")
             plt.ylabel("True Positive Rate (Recall)")
```

```
In [53]: def plot_roc(x,y,w):
             fpr, tpr, auc_thresholds = roc_curve(x, y)
             plot_roc_curve(fpr, tpr, w)
```

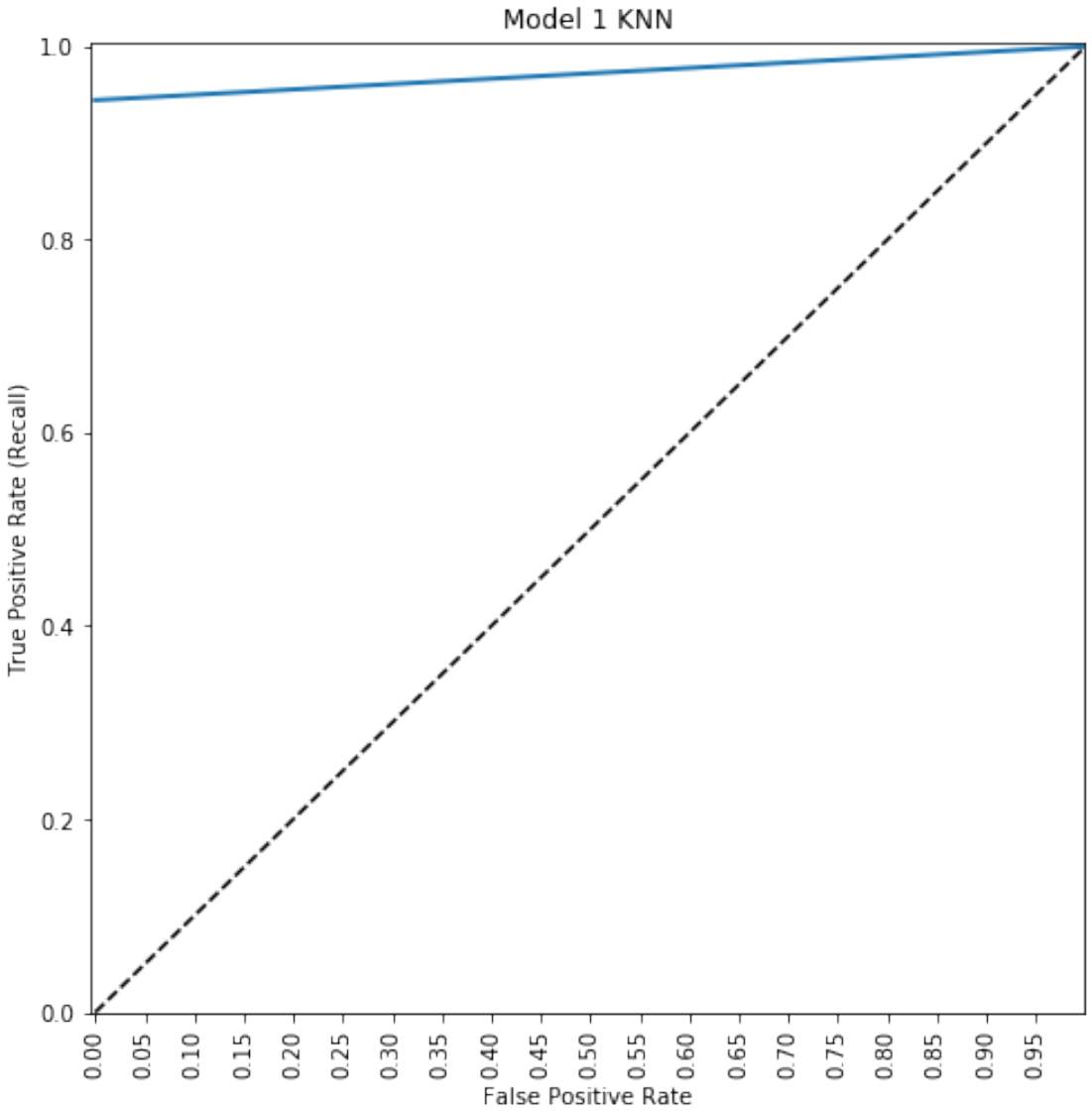
```
In [54]: def plot_met(x,y):
             cm_all = confusion_matrix(x, y)
             sns.heatmap(cm_all, annot = True, fmt = "d")
             plt.xlabel('Predicted')
             plt.ylabel('Actual')
             print(classification_report(x, y))
             print("Accuracy:", accuracy_score(x, y))
```

```
In [55]: plot_met(y_val, y_pred_all)
```

```
             precision    recall  f1-score   support

        0.0       0.96      1.00      0.98        55
        1.0       1.00      0.94      0.97        36

avg / total       0.98      0.98      0.98        91

Accuracy: 0.978021978021978
```




```
In [56]: plot_roc(y_val, y_pred_all, "Model 1 KNN")
```



1.0.1 Model 2

```
In [57]: knn_grid(X_train[cor_features], y_train)
```

```
Out[57]: ({'n_neighbors': 3}, 0.945054945054945)
```

```
In [58]: knn_cor = KNeighborsClassifier(n_neighbors=3)
         knn_cor.fit(X_train[cor_features], y_train)
```

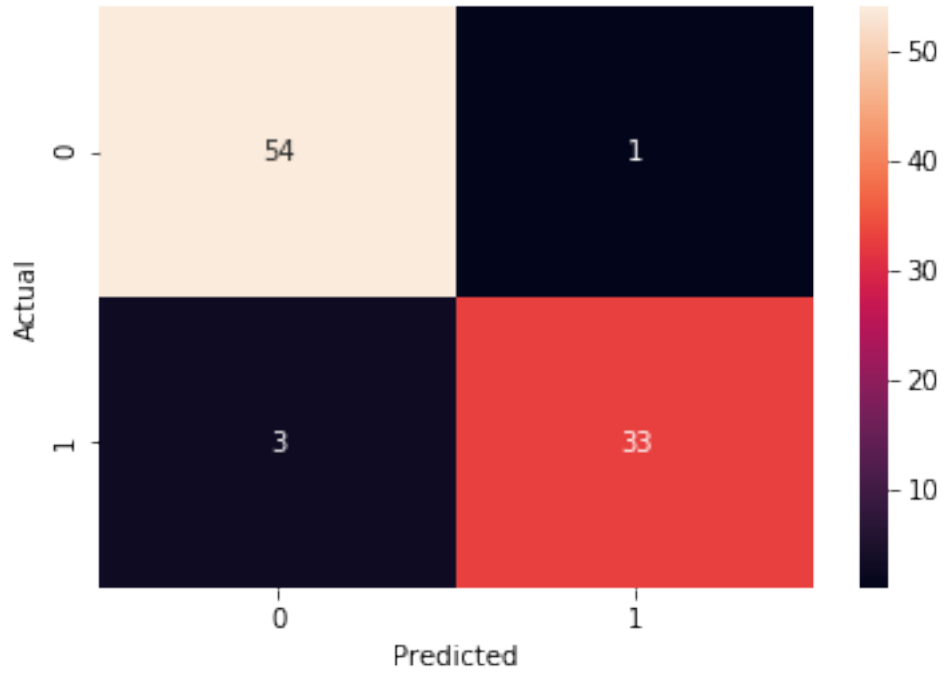
```
Out[58]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                    weights='uniform')
```

```
In [59]: y_pred_cor = knn_cor.predict(X_val[cor_features])
```

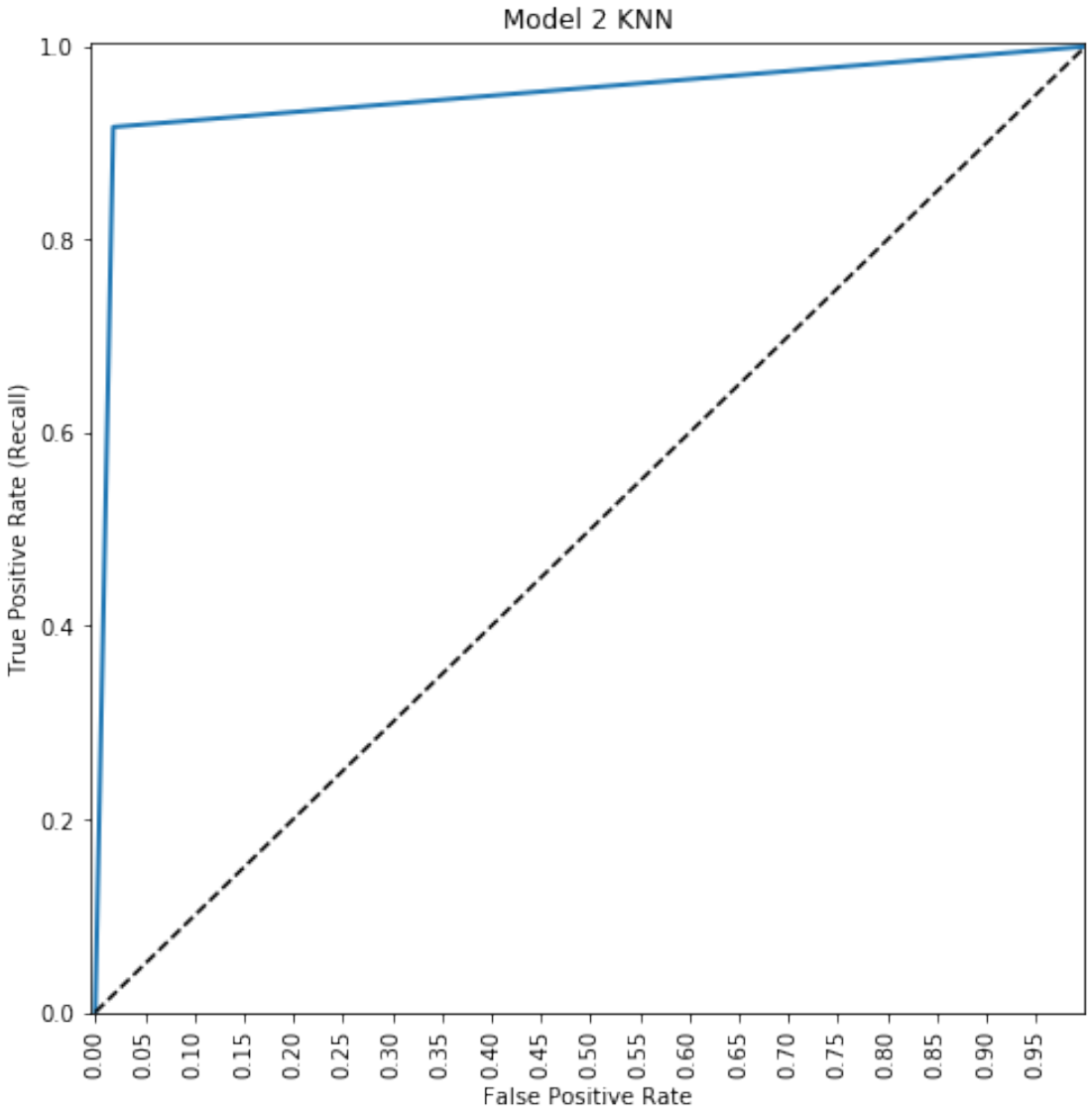
In [60]: plot_met(y_val,y_pred_cor)

	precision	recall	f1-score	support
0.0	0.95	0.98	0.96	55
1.0	0.97	0.92	0.94	36
avg / total	0.96	0.96	0.96	91

Accuracy: 0.9560439560439561



In [61]: plot_roc(y_val, y_pred_cor, "Model 2 KNN")



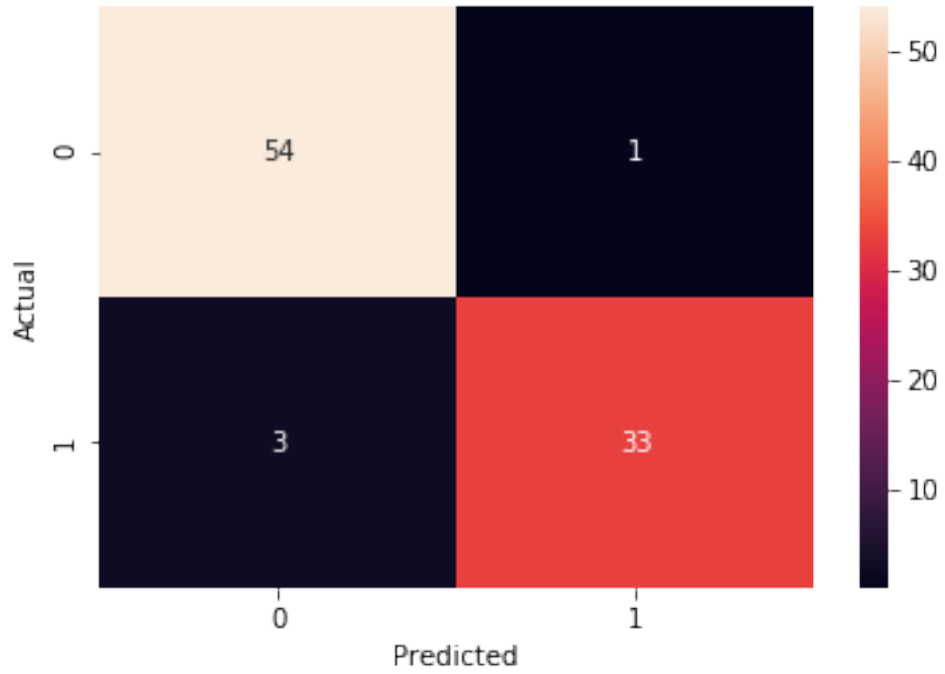
Model 3

```
In [62]: knn_grid(X_train[rf_features], y_train)

Out[62]: ({'n_neighbors': 13}, 0.9560439560439561)

In [63]: knn_rf = KNeighborsClassifier(n_neighbors=13)
         knn_rf.fit(X_train[rf_features], y_train)

Out[63]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=13, p=2,
                    weights='uniform')

In [64]: y_pred_rf = knn_rf.predict(X_val[rf_features])
```

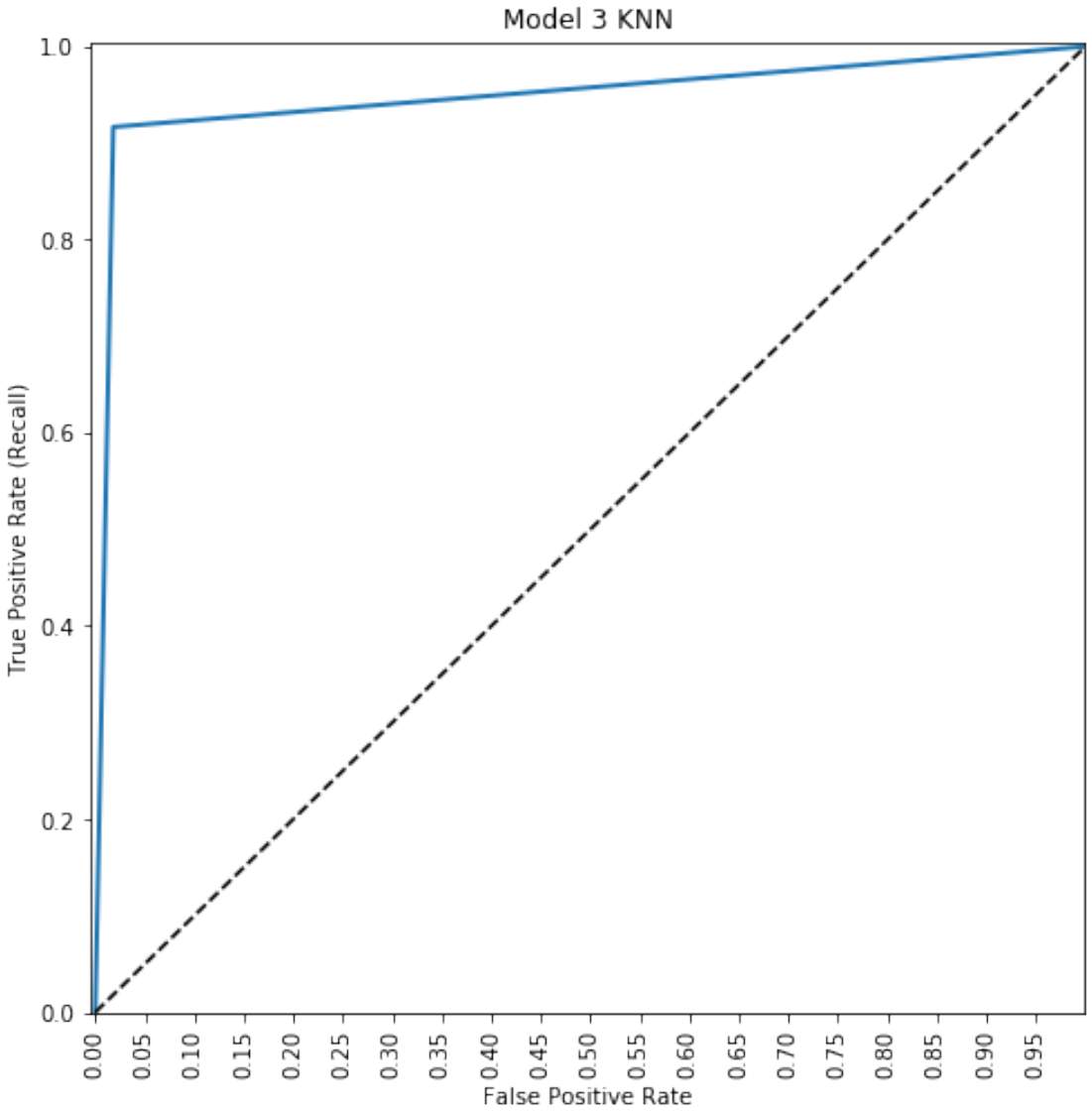
In [65]: plot_met(y_val, y_pred_rf)

	precision	recall	f1-score	support
0.0	0.95	0.98	0.96	55
1.0	0.97	0.92	0.94	36
avg / total	0.96	0.96	0.96	91

Accuracy: 0.9560439560439561



In [66]: plot_roc(y_val, y_pred_rf, "Model 3 KNN")



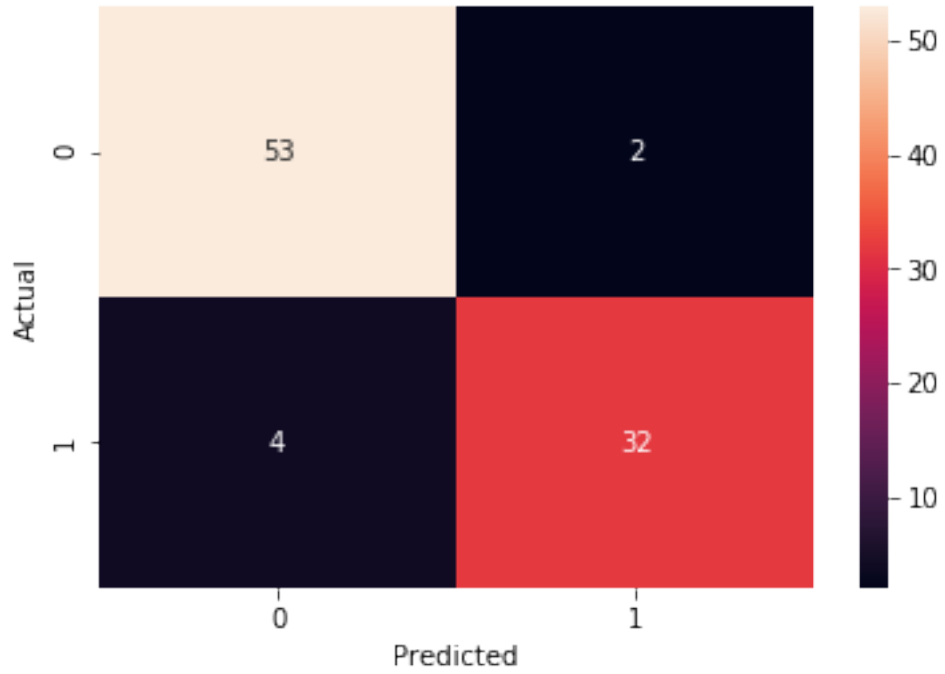
Model 4

```
In [67]: knn_grid(X_train[eda_features], y_train)

Out[67]: ({'n_neighbors': 6}, 0.945054945054945)

In [68]: knn_eda = KNeighborsClassifier(n_neighbors=6)
         knn_eda.fit(X_train[eda_features], y_train)

Out[68]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=6, p=2,
                    weights='uniform')

In [69]: y_pred_eda = knn_eda.predict(X_val[eda_features])
```

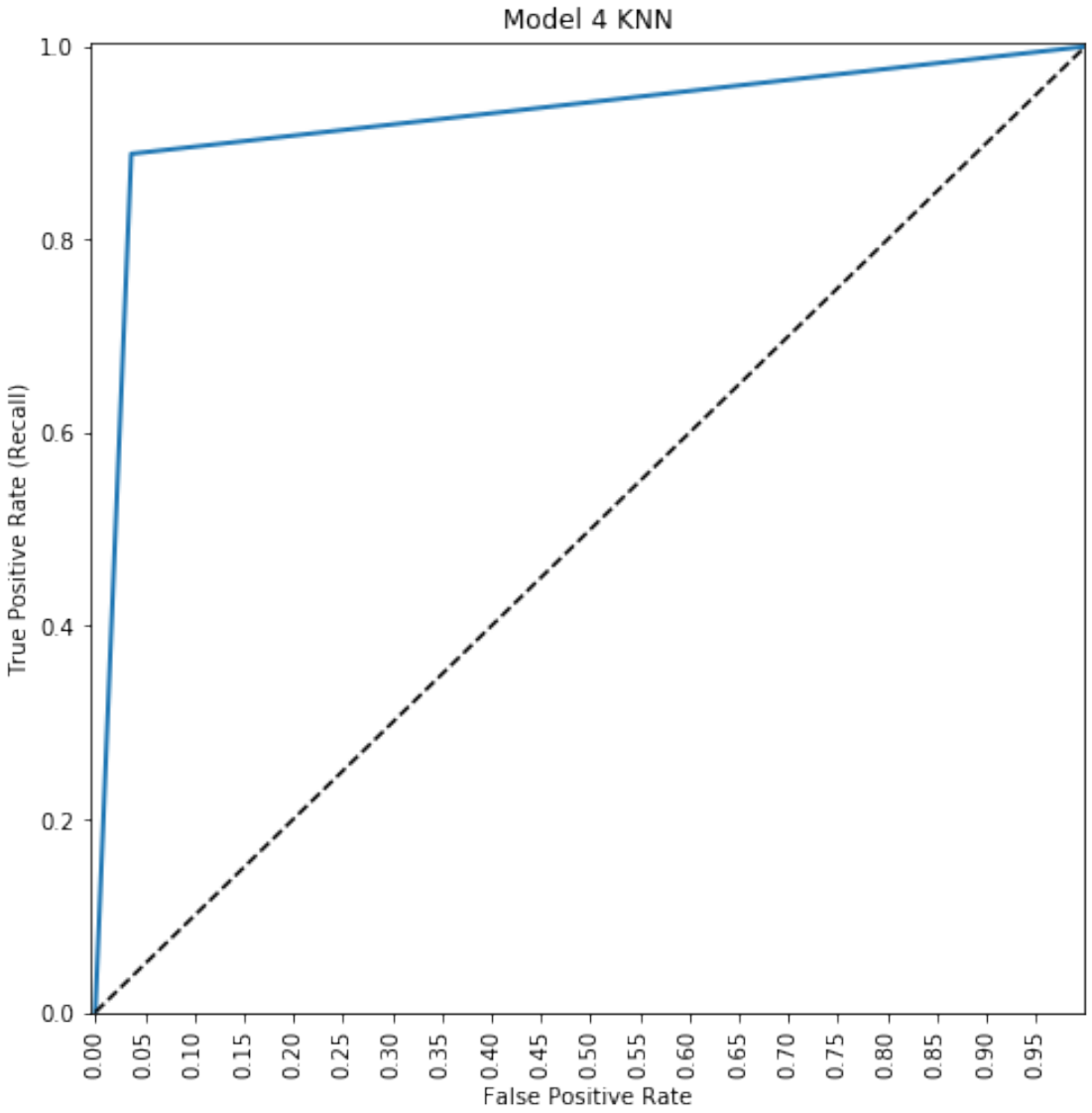
In [70]: plot_met(y_val, y_pred_eda)

	precision	recall	f1-score	support
0.0	0.93	0.96	0.95	55
1.0	0.94	0.89	0.91	36
avg / total	0.93	0.93	0.93	91

Accuracy: 0.9340659340659341



In [71]: plot_roc(y_val, y_pred_eda, "Model 4 KNN")



In [72]:
```
from sklearn.metrics import precision_score, recall_score, f1_score, auc, roc_auc_sco
```

In [73]:
```
def scores(x,y):
    acc = accuracy_score(x,y)
    pre = precision_score(x,y)
    rec = recall_score(x,y)
    f1 = f1_score(x,y)
    auc2 = roc_auc_score(x,y)
    print("Accuracy:",round(acc,4))
    print("Precision:", round(pre,4))
    print("Recall:", round(rec,4))
    print("F1 Score:",round(f1,4))
    print("AUC:", round(auc2,4))
```

```
In [74]: print("Model 1 KNN")
         scores(y_val, y_pred_all)
         print("--"*40)
         print("Model 2 KNN")
         scores(y_val, y_pred_cor)
         print("--"*40)
         print("Model 3 KNN")
         scores(y_val, y_pred_rf)
         print("--"*40)
         print("Model 4 KNN")
         scores(y_val, y_pred_eda)
```

```
Model 1 KNN
Accuracy: 0.978
Precision: 1.0
Recall: 0.9444
F1 Score: 0.9714
AUC: 0.9722
--------------------------------------------------------------------------------
Model 2 KNN
Accuracy: 0.956
Precision: 0.9706
Recall: 0.9167
F1 Score: 0.9429
AUC: 0.9492
--------------------------------------------------------------------------------
Model 3 KNN
Accuracy: 0.956
Precision: 0.9706
Recall: 0.9167
F1 Score: 0.9429
AUC: 0.9492
--------------------------------------------------------------------------------
Model 4 KNN
Accuracy: 0.9341
Precision: 0.9412
Recall: 0.8889
F1 Score: 0.9143
AUC: 0.9263
```

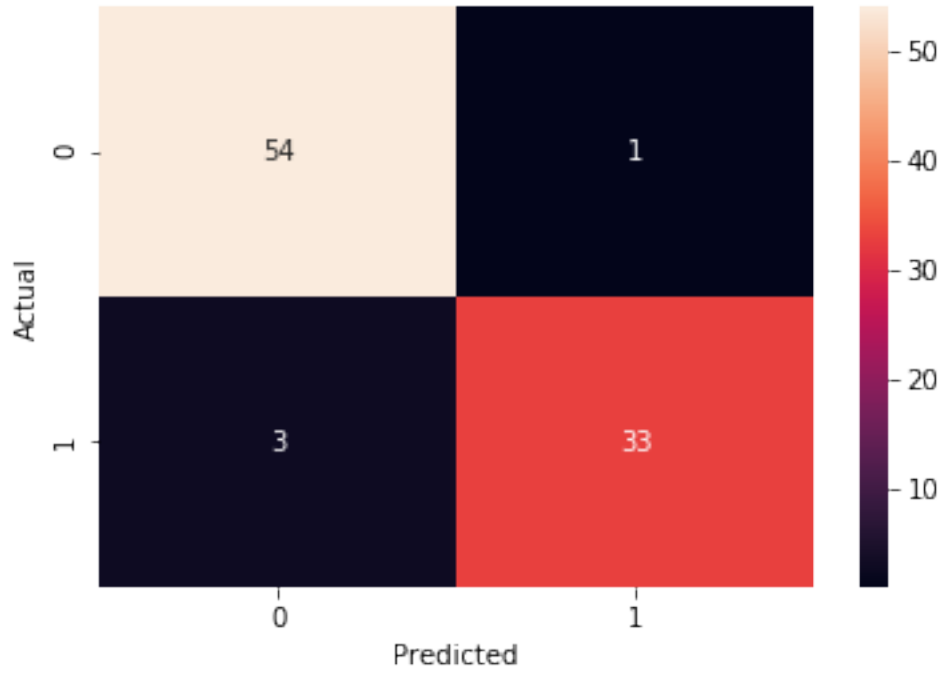
1.0.2 Logistic Regression

```
In [75]: from sklearn.linear_model import LogisticRegression
```

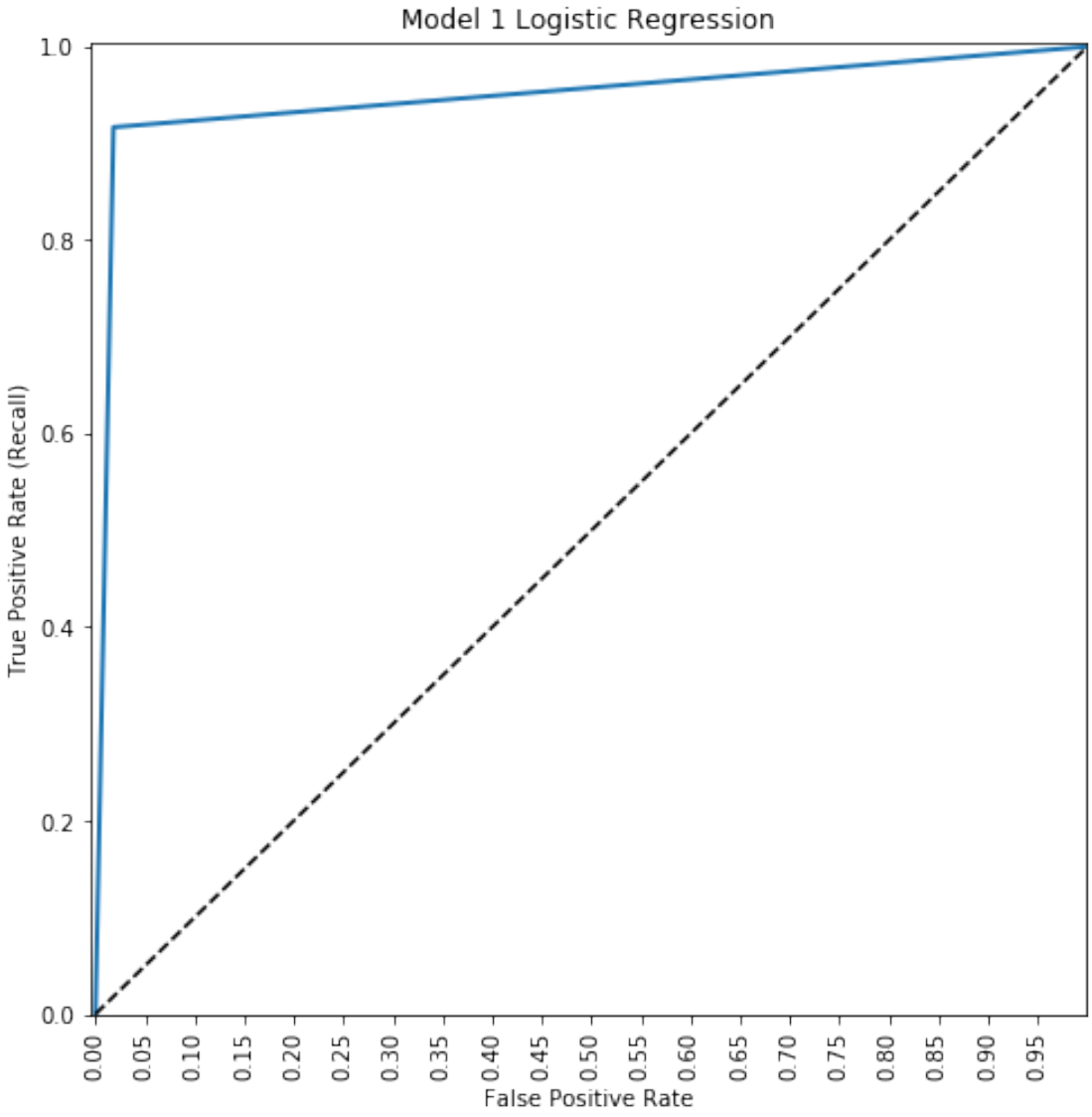
Model 1

```
In [76]: log_all = LogisticRegression()
         log_all.fit(X_train, y_train)
```

```
Out[76]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                    verbose=0, warm_start=False)

In [77]: pred_all = log_all.predict(X_val)

In [78]: plot_met(y_val, pred_all)

             precision    recall  f1-score   support

        0.0       0.95      0.98      0.96        55
        1.0       0.97      0.92      0.94        36

avg / total       0.96      0.96      0.96        91

Accuracy: 0.9560439560439561
```



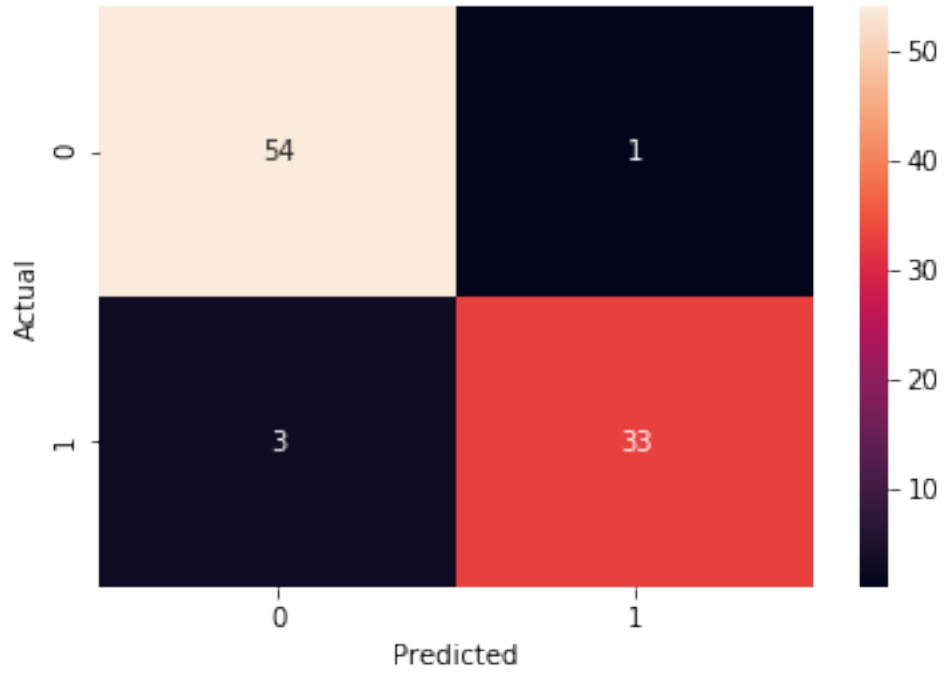

```
In [79]: plot_roc(y_val, pred_all, "Model 1 Logistic Regression")
```



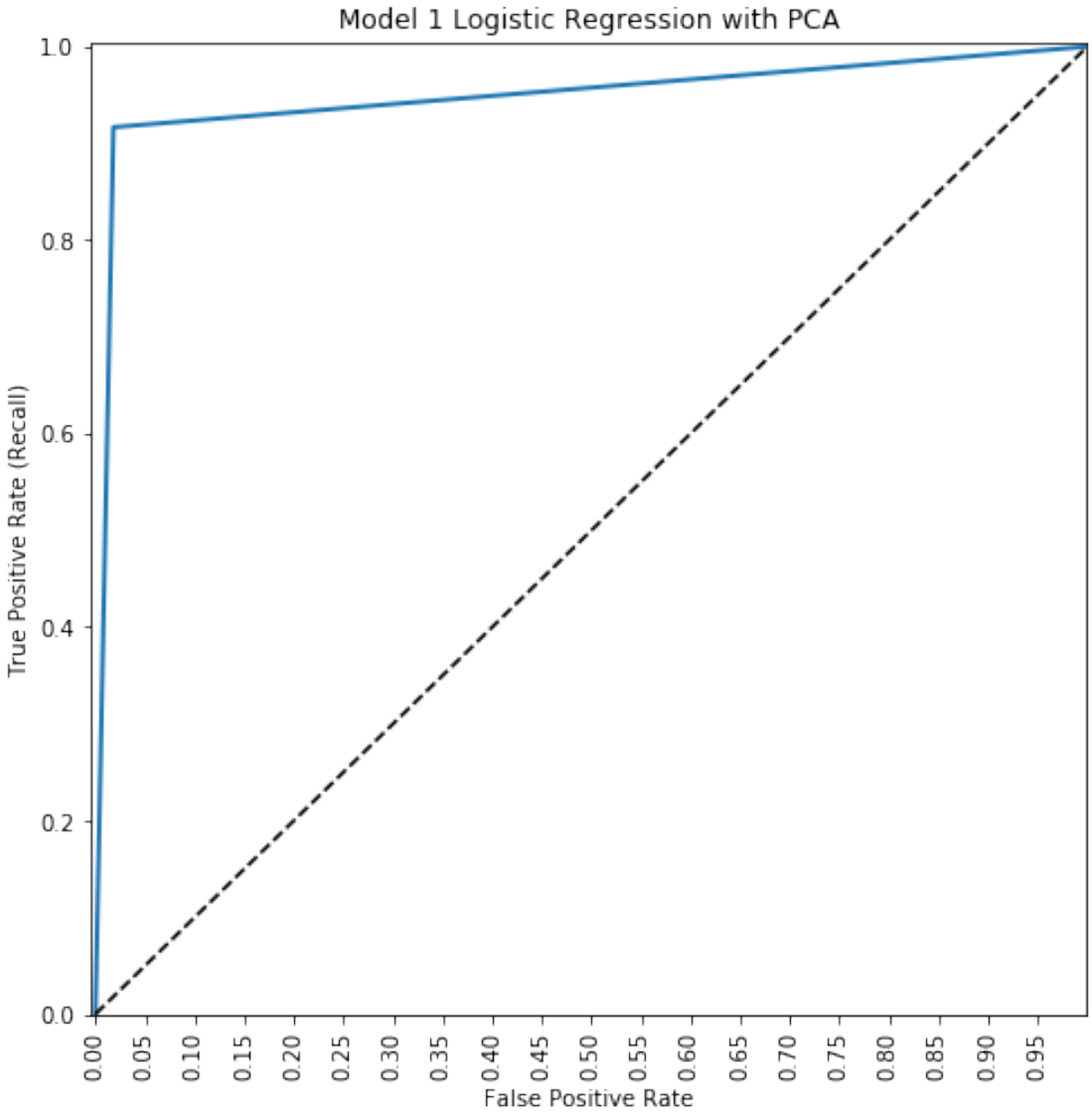
Model 1 with PCA

```
In [80]: from sklearn.decomposition import PCA

In [81]: pca = PCA(.95)

In [82]: X_train_pca = pca.fit_transform(X_train)
         X_val_pca = pca.transform(X_val)
         X_test_pca = pca.transform(X_test)

In [83]: log_pca = LogisticRegression()
         log_pca.fit(X_train_pca, y_train)
```

```
Out[83]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                    verbose=0, warm_start=False)

In [84]: pred_pca = log_pca.predict(X_val_pca)

In [85]: plot_met(y_val, pred_pca)

             precision    recall  f1-score   support

        0.0       0.95      0.98      0.96        55
        1.0       0.97      0.92      0.94        36

avg / total       0.96      0.96      0.96        91

Accuracy: 0.9560439560439561
```



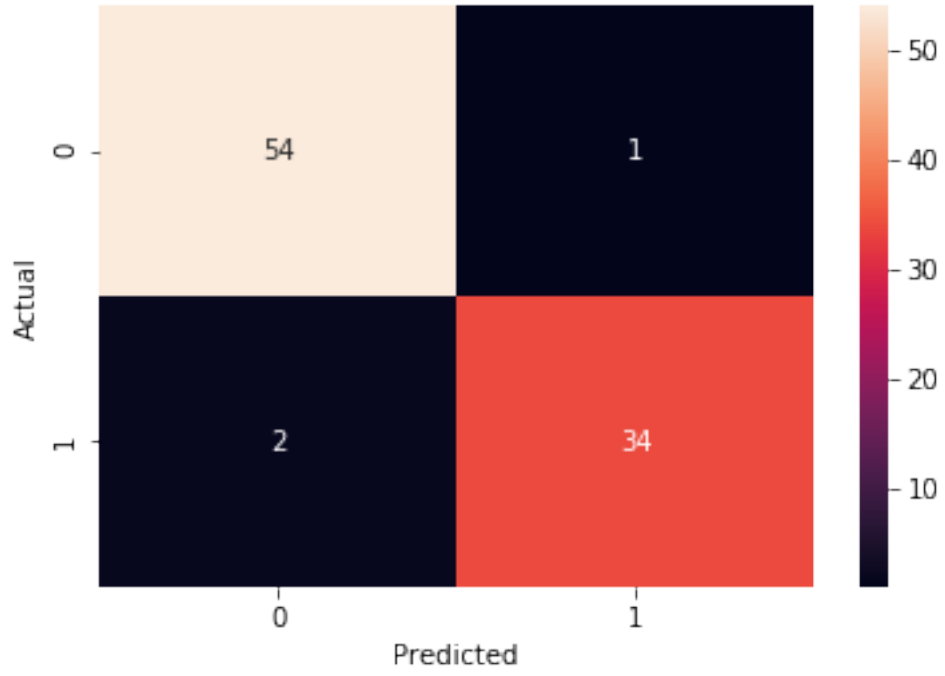

```
In [86]: plot_roc(y_val, pred_pca, "Model 1 Logistic Regression with PCA")
```



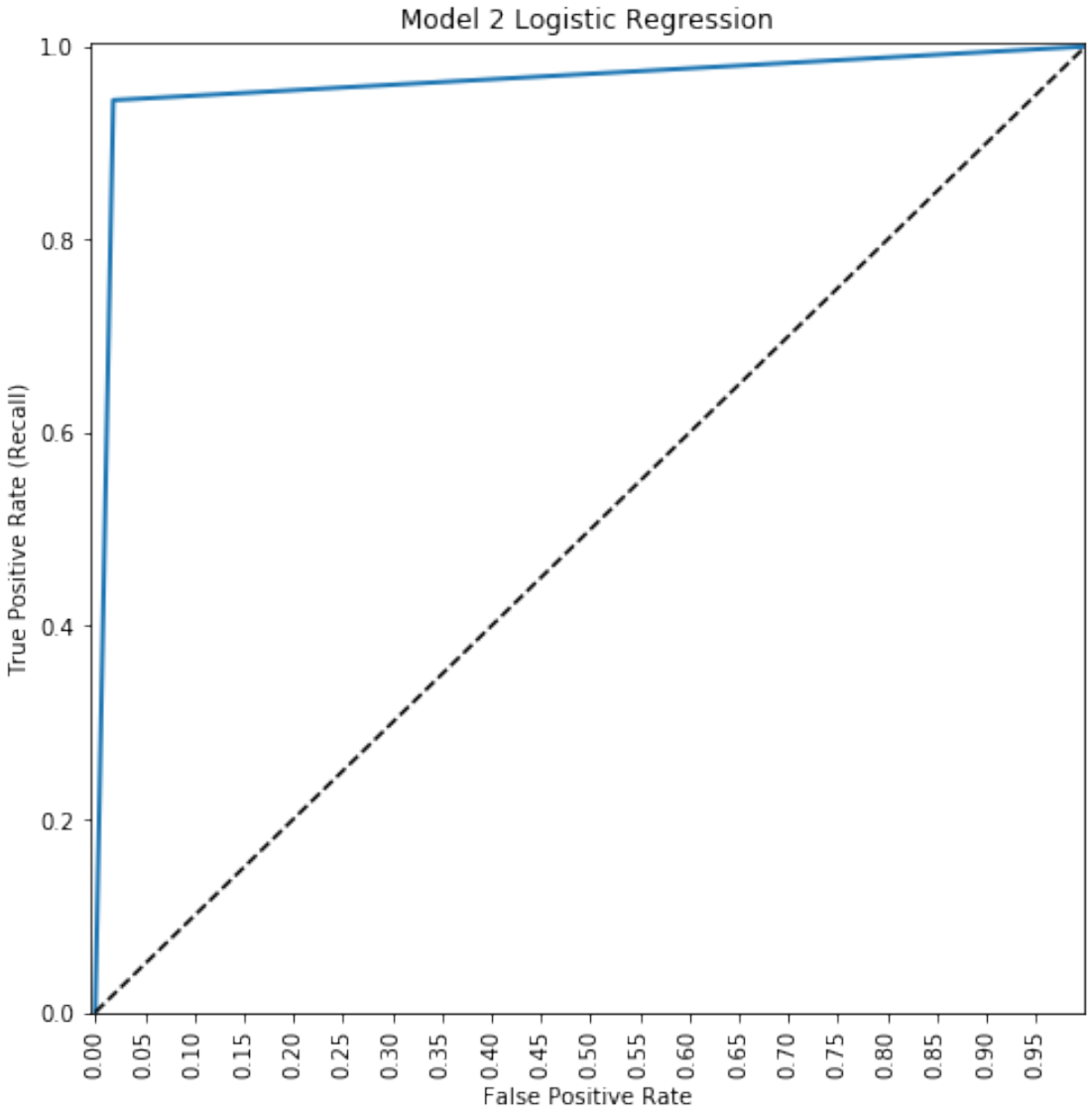
Model 2

```
In [87]: log_cor = LogisticRegression(penalty='l1')
         log_cor.fit(X_train[cor_features], y_train)

Out[87]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [88]: pred_cor = log_cor.predict(X_val[cor_features])

In [89]: plot_met(y_val, pred_cor)
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.98      0.97        55
         1.0       0.97      0.94      0.96        36

avg / total       0.97      0.97      0.97        91

Accuracy: 0.967032967032967
```



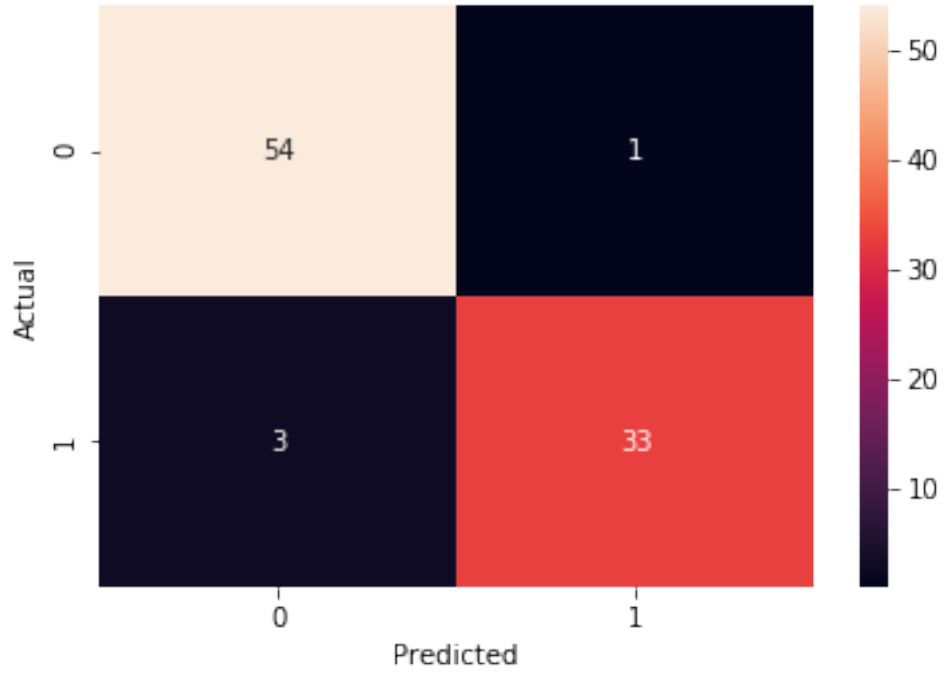

```
In [90]: plot_roc(y_val, pred_cor, "Model 2 Logistic Regression")
```



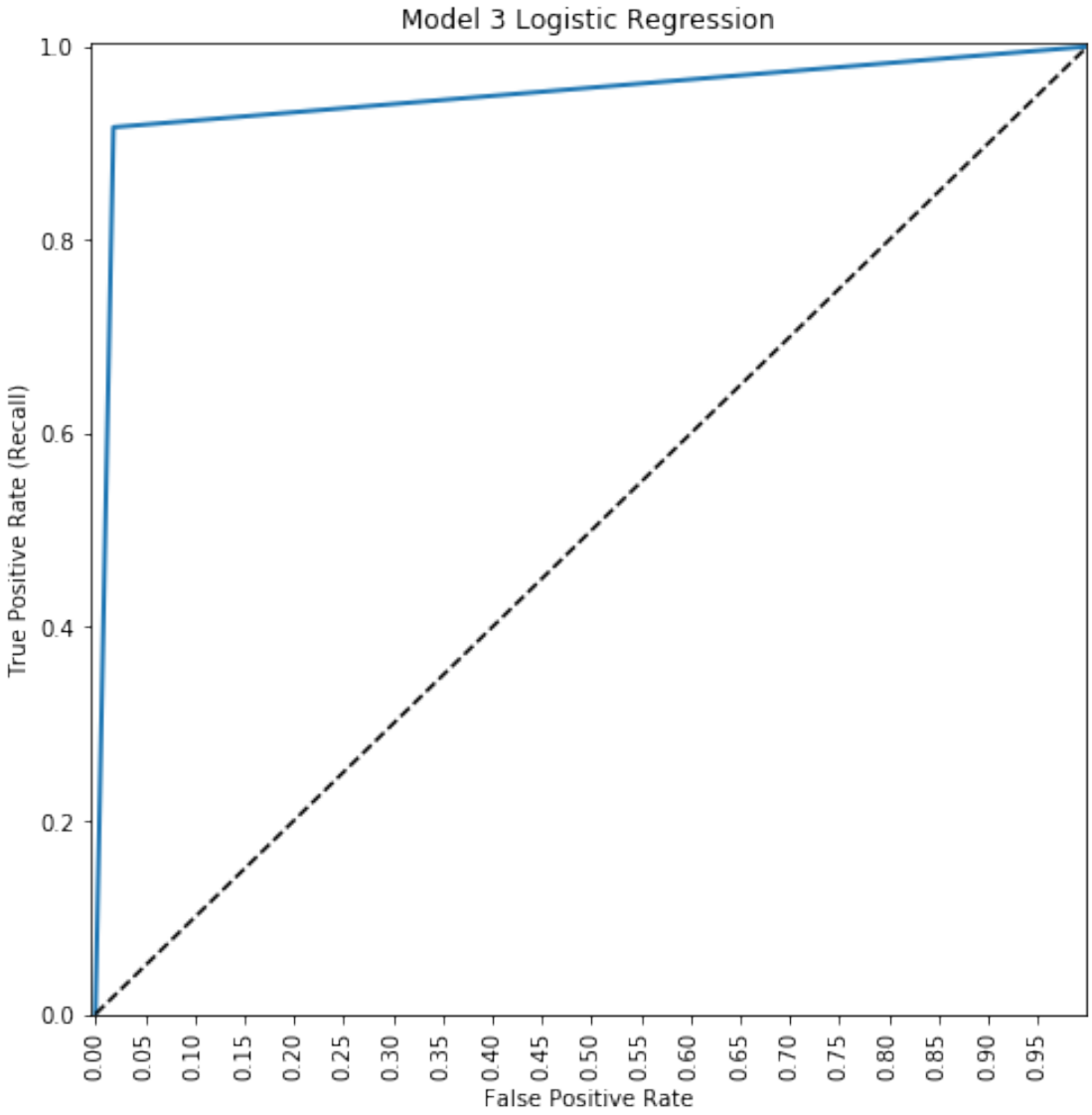
Model 3

```
In [91]: log_rf = LogisticRegression()
         log_rf.fit(X_train[rf_features], y_train)

Out[91]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [92]: pred_rf = log_rf.predict(X_val[rf_features])

In [93]: plot_met(y_val, pred_rf)
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.98      0.96        55
         1.0       0.97      0.92      0.94        36

avg / total       0.96      0.96      0.96        91

Accuracy: 0.9560439560439561
```



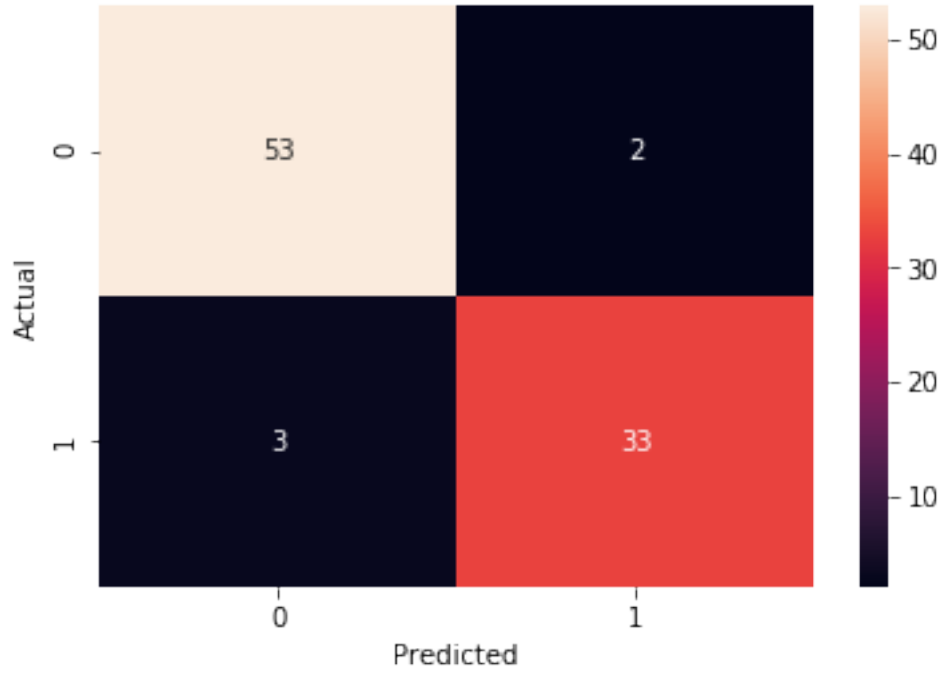
In [94]: plot_roc(y_val, pred_rf, "Model 3 Logistic Regression")



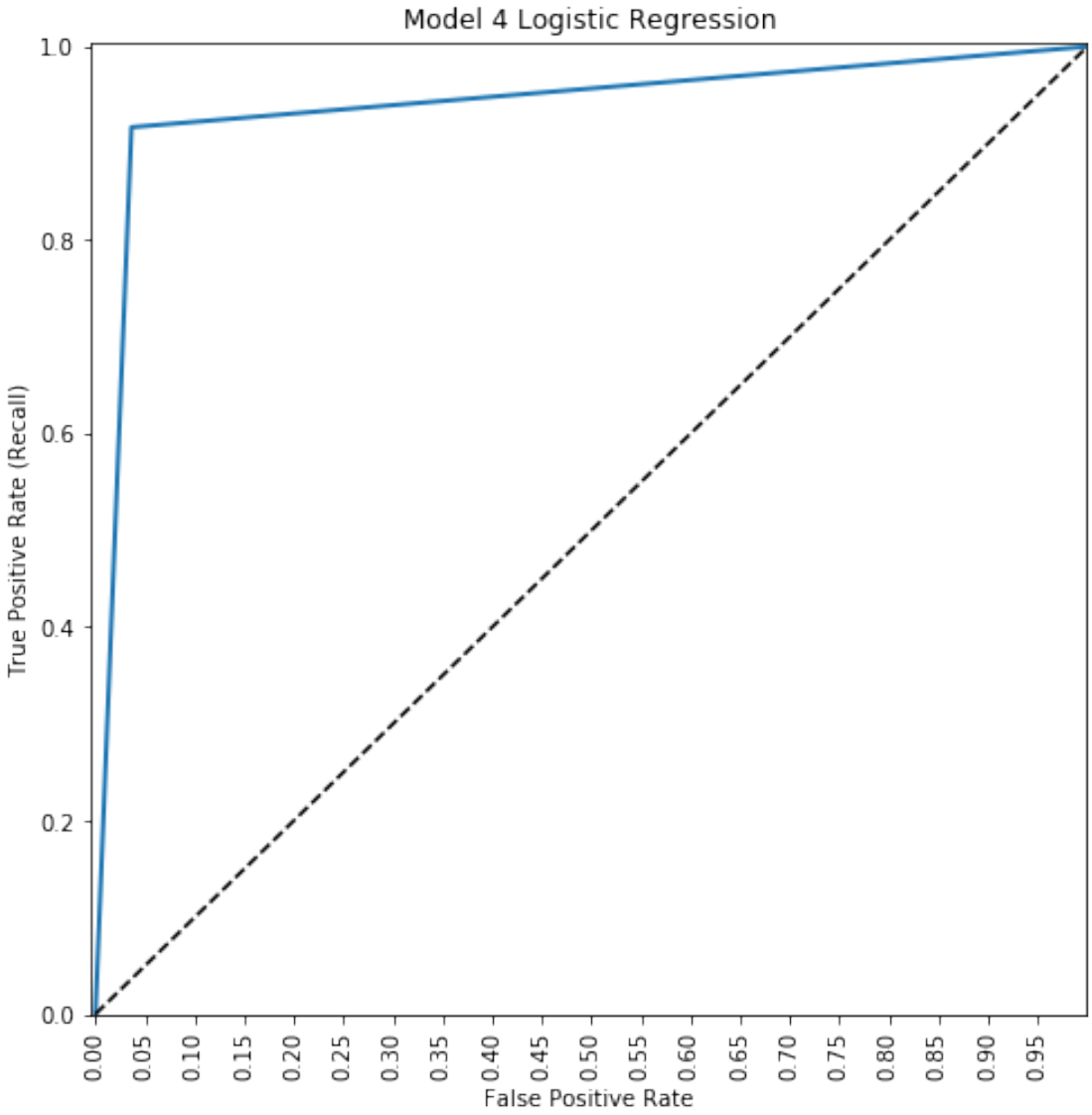
Model 4

```
In [95]: log_eda = LogisticRegression(penalty='l1')
         log_eda.fit(X_train[eda_features], y_train)

Out[95]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [96]: pred_eda = log_eda.predict(X_val[eda_features])

In [97]: plot_met(y_val, pred_eda)
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.96      0.95        55
         1.0       0.94      0.92      0.93        36

avg / total       0.95      0.95      0.94        91

Accuracy: 0.945054945054945
```



In [98]: `plot_roc(y_val, pred_eda, "Model 4 Logistic Regression")`




```
In [99]: print("Model 1 Logistic Regression")
         scores(y_val, pred_all)
         print("--"*40)
         print("Model 1 Logistic Regression with PCA")
         scores(y_val, pred_pca)
         print("--"*40)
         print("Model 2 Logistic Regression")
         scores(y_val, pred_cor)
         print("--"*40)
         print("Model 3 Logistic Regression")
         scores(y_val, pred_rf)
         print("--"*40)
```

```
print("Model 4 Logistic Regression")
scores(y_val, pred_eda)
```

```
Model 1 Logistic Regression
Accuracy: 0.956
Precision: 0.9706
Recall: 0.9167
F1 Score: 0.9429
AUC: 0.9492
--------------------------------------------------------------------------------
Model 1 Logistic Regression with PCA
Accuracy: 0.956
Precision: 0.9706
Recall: 0.9167
F1 Score: 0.9429
AUC: 0.9492
--------------------------------------------------------------------------------
Model 2 Logistic Regression
Accuracy: 0.967
Precision: 0.9714
Recall: 0.9444
F1 Score: 0.9577
AUC: 0.9631
--------------------------------------------------------------------------------
Model 3 Logistic Regression
Accuracy: 0.956
Precision: 0.9706
Recall: 0.9167
F1 Score: 0.9429
AUC: 0.9492
--------------------------------------------------------------------------------
Model 4 Logistic Regression
Accuracy: 0.9451
Precision: 0.9429
Recall: 0.9167
F1 Score: 0.9296
AUC: 0.9402
```

LDA

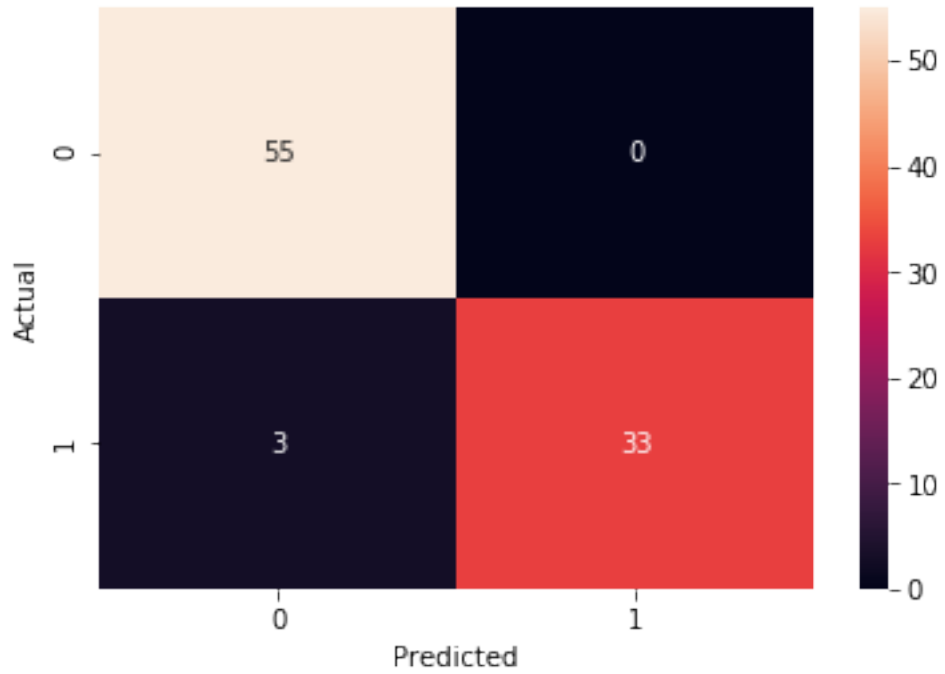
```
In [100]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

Model 1

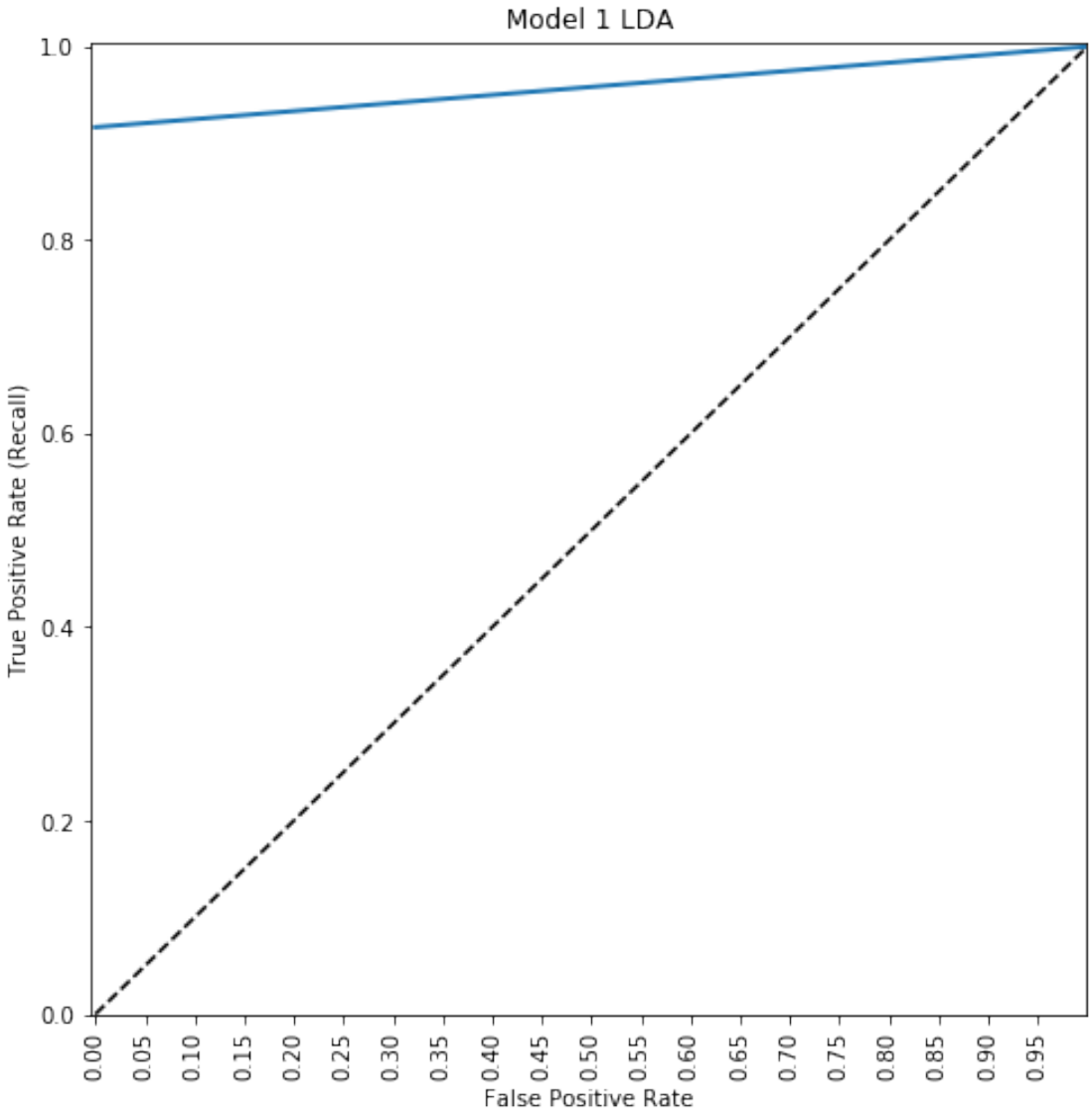
```
In [101]: lda_all = LDA()
```

```
In [102]: lda_all.fit(X_train, y_train)
```

```
Out[102]: LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                                     solver='svd', store_covariance=False, tol=0.0001)

In [103]: lda_pred_all = lda_all.predict(X_val)

In [104]: plot_met(y_val, lda_pred_all)

              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97        55
         1.0       1.00      0.92      0.96        36

avg / total       0.97      0.97      0.97        91

Accuracy: 0.967032967032967
```



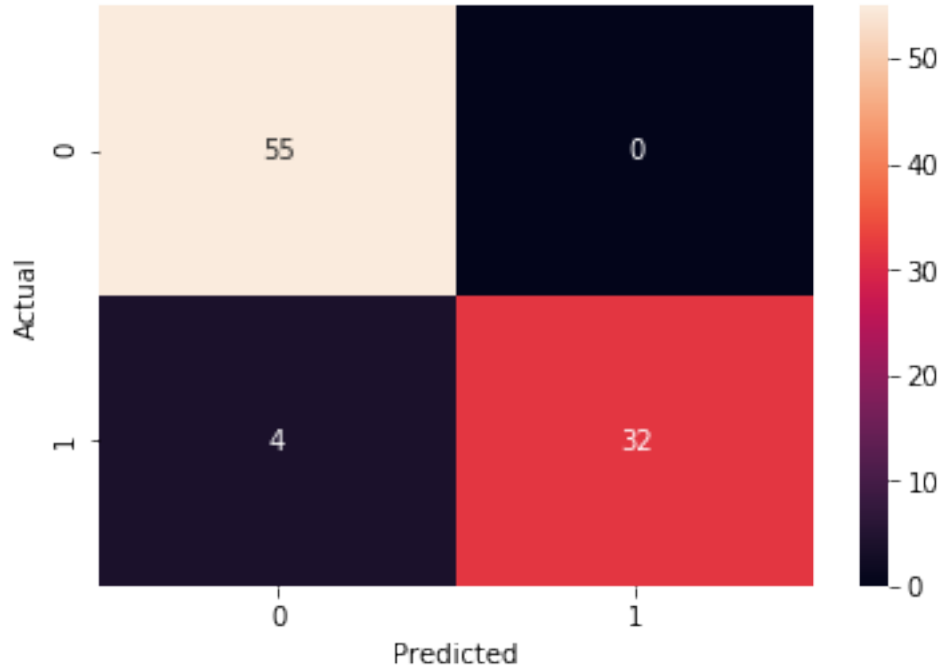

```
In [105]: plot_roc(y_val, lda_pred_all, "Model 1 LDA")
```



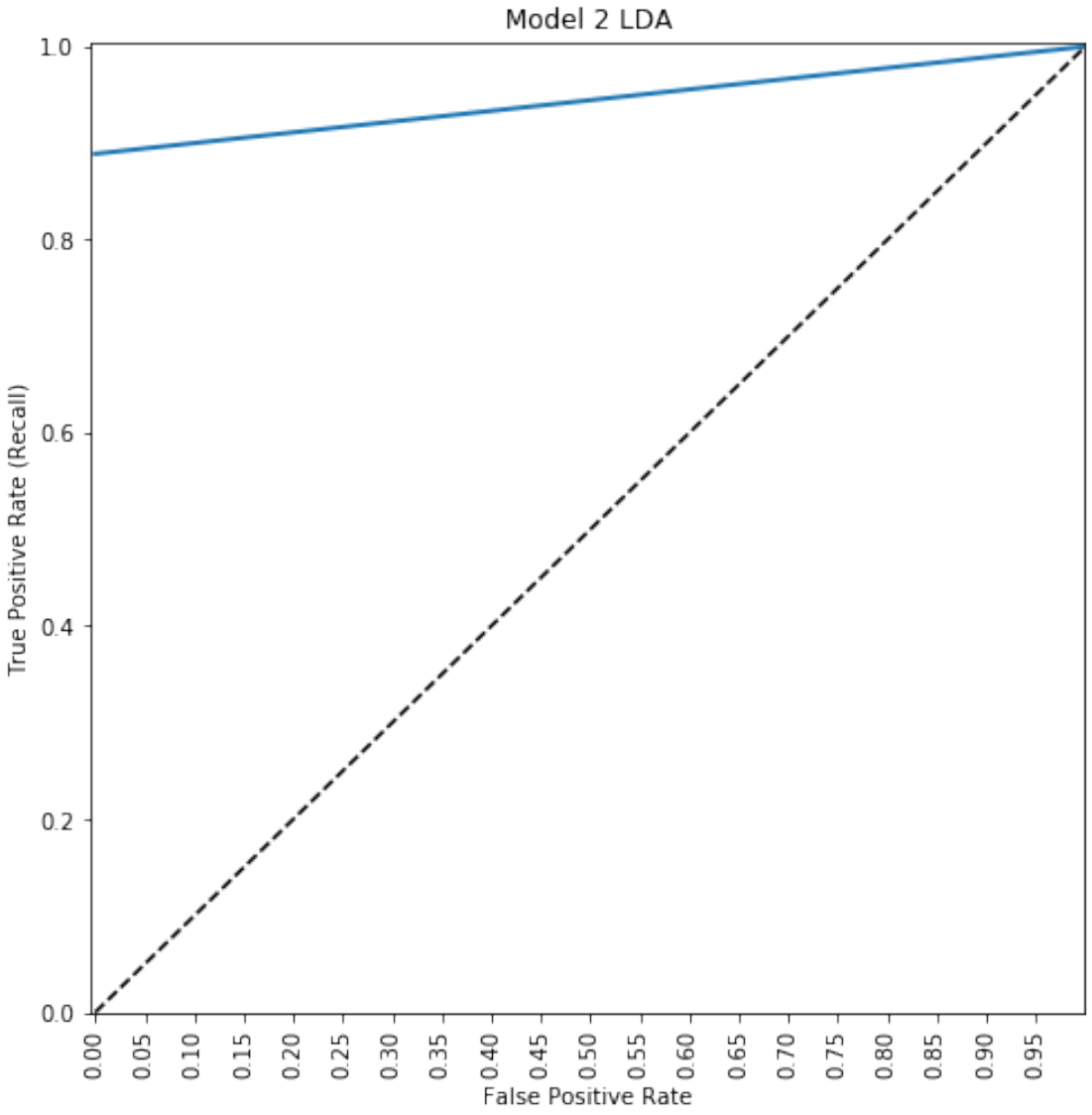
Model 2

```
In [106]: lda_cor = LDA()
          lda_cor.fit(X_train[cor_features], y_train)

Out[106]: LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                        solver='svd', store_covariance=False, tol=0.0001)

In [107]: lda_pred_cor = lda_cor.predict(X_val[cor_features])

In [108]: plot_met(y_val, lda_pred_cor)

              precision    recall  f1-score   support
```

```
          0.0       0.93      1.00      0.96        55
          1.0       1.00      0.89      0.94        36

avg / total       0.96      0.96      0.96        91

Accuracy: 0.9560439560439561
```



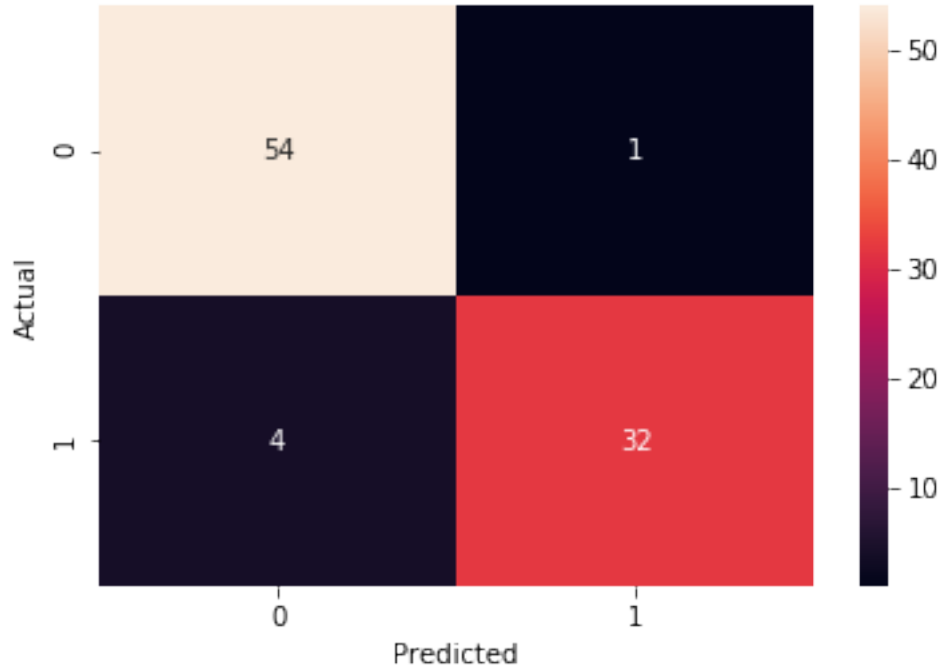

```
In [109]: plot_roc(y_val, lda_pred_cor, "Model 2 LDA")
```



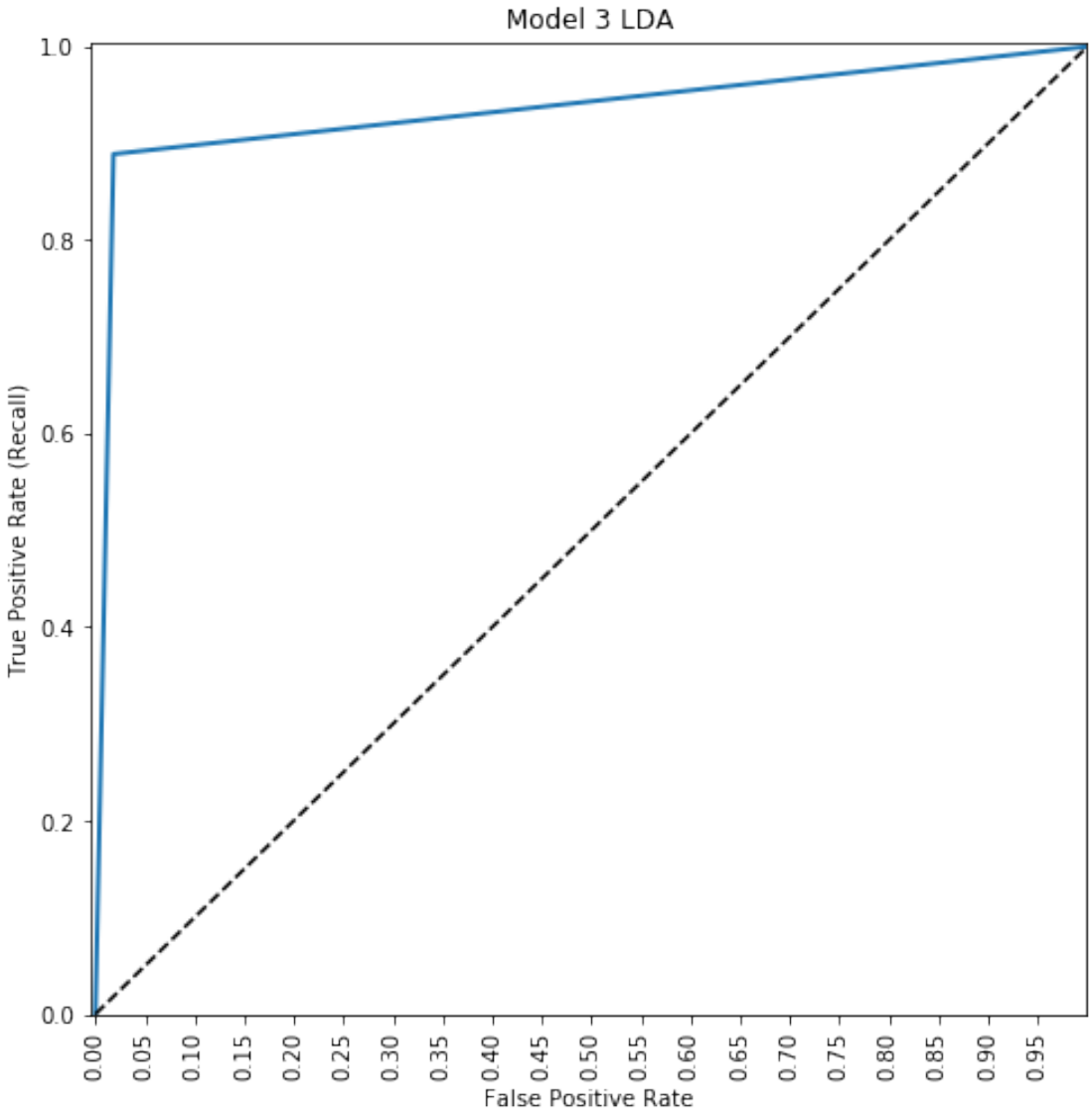
Model 3

```
In [110]: lda_rf = LDA()
          lda_rf.fit(X_train[rf_features], y_train)

Out[110]: LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                        solver='svd', store_covariance=False, tol=0.0001)

In [111]: lda_pred_rf = lda_rf.predict(X_val[rf_features])

In [112]: plot_met(y_val, lda_pred_rf)

              precision    recall  f1-score   support
```

```
          0.0       0.93      0.98      0.96        55
          1.0       0.97      0.89      0.93        36

avg / total       0.95      0.95      0.94        91

Accuracy: 0.945054945054945
```



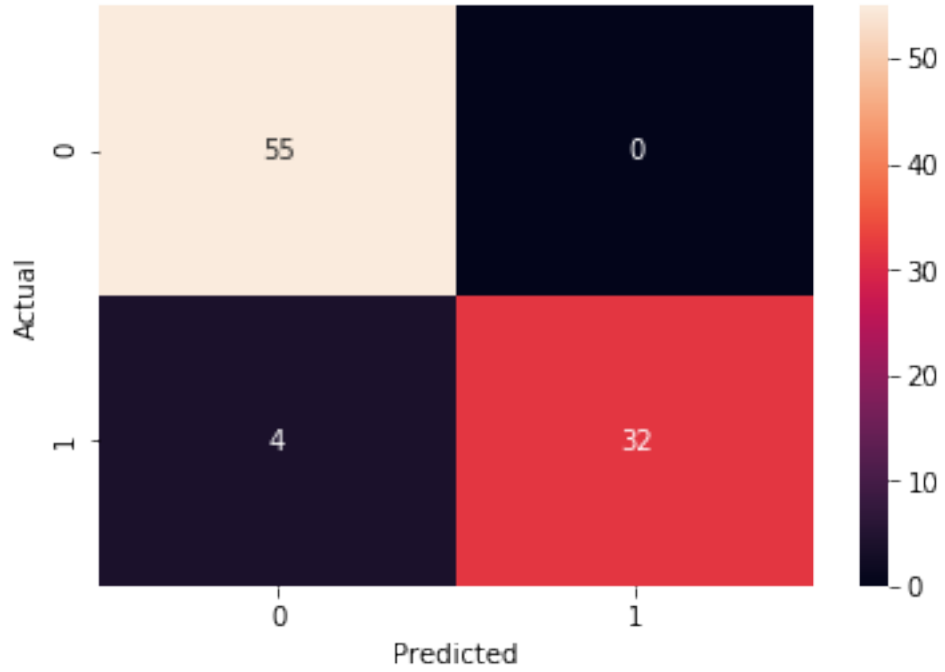

```
In [113]: plot_roc(y_val, lda_pred_rf, "Model 3 LDA")
```



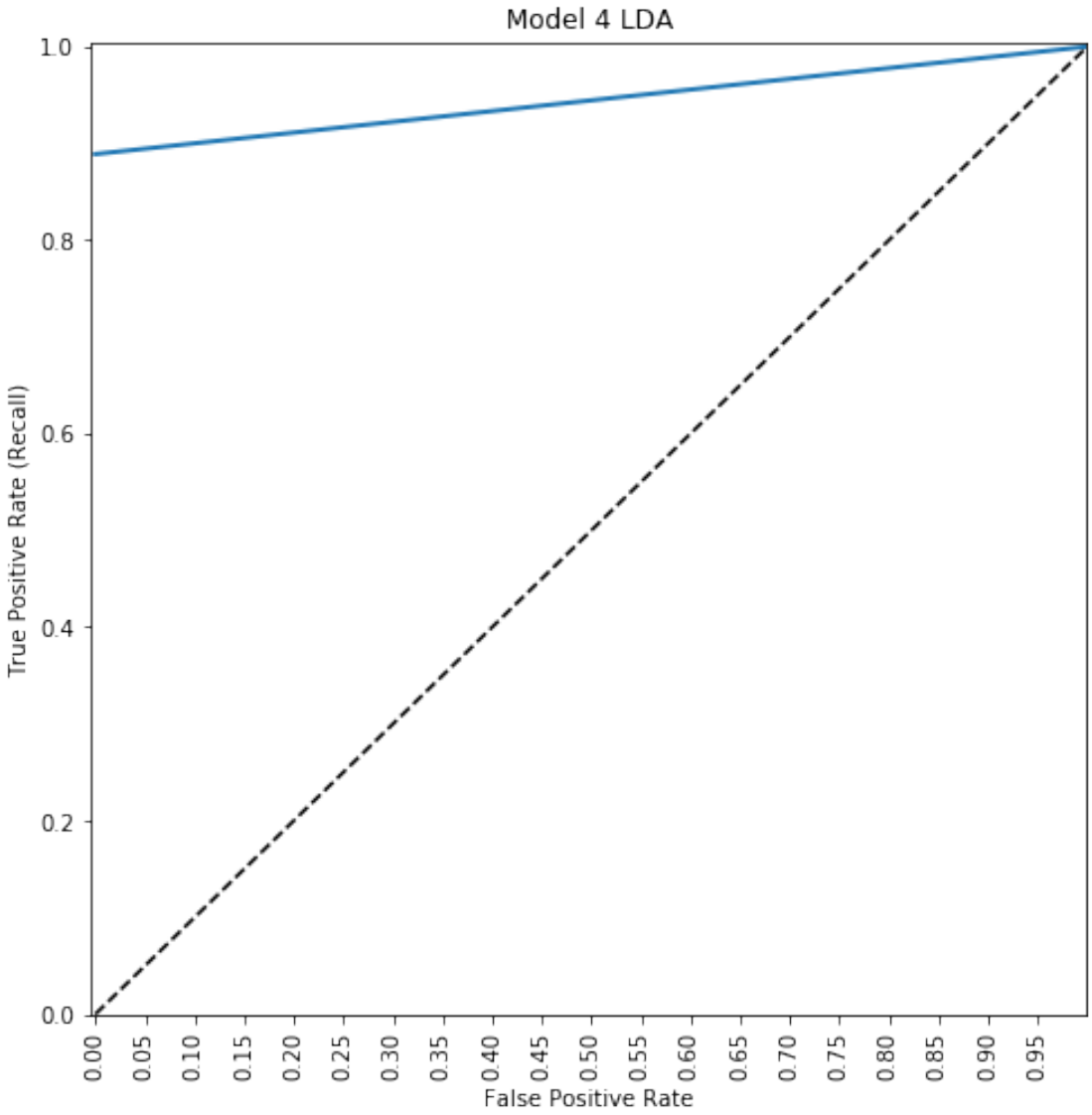
Model 4

```
In [114]: lda_eda = LDA()
          lda_eda.fit(X_train[eda_features], y_train)

Out[114]: LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                        solver='svd', store_covariance=False, tol=0.0001)

In [115]: lda_pred_eda = lda_eda.predict(X_val[eda_features])

In [116]: plot_met(y_val, lda_pred_eda)

              precision    recall  f1-score   support
```

```
          0.0       0.93      1.00      0.96        55
          1.0       1.00      0.89      0.94        36

avg / total       0.96      0.96      0.96        91

Accuracy: 0.9560439560439561
```



In [117]: `plot_roc(y_val, lda_pred_eda, "Model 4 LDA")`




```
In [118]: print("Model 1 LDA")
          scores(y_val, lda_pred_all)
          print("--"*40)
          print("Model 2 LDA")
          scores(y_val,lda_pred_cor)
          print("--"*40)
          print("Model 3 LDA")
          scores(y_val, lda_pred_rf)
          print("--"*40)
          print("Model 4 LDA")
          scores(y_val, lda_pred_eda)
```

```
Model 1 LDA
```

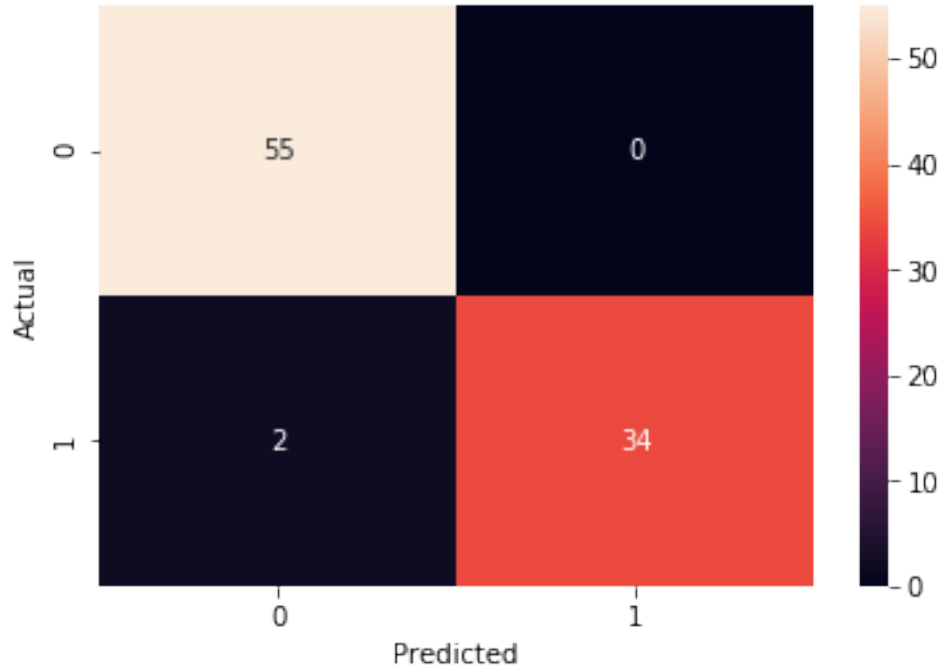
```
Accuracy: 0.967
Precision: 1.0
Recall: 0.9167
F1 Score: 0.9565
AUC: 0.9583
--------------------------------------------------------------------------------
Model 2 LDA
Accuracy: 0.956
Precision: 1.0
Recall: 0.8889
F1 Score: 0.9412
AUC: 0.9444
--------------------------------------------------------------------------------
Model 3 LDA
Accuracy: 0.9451
Precision: 0.9697
Recall: 0.8889
F1 Score: 0.9275
AUC: 0.9354
--------------------------------------------------------------------------------
Model 4 LDA
Accuracy: 0.956
Precision: 1.0
Recall: 0.8889
F1 Score: 0.9412
AUC: 0.9444
```

1.0.3 Best Models

Based on all the information from the models I have narrowed it down to two models, Model 1 using KNN and Model 2 using Logistic Regression. When working with cancer classification we are looking for models that have a small amount of false negatives. In our problem a false negative is predicting benign (0) and having the actual outcome be malignant(1). Another way to test if a model handles false negative outcomes well is to look at the recall score, which is the percentage of patients that actually have cancer that the model predicted to have cancer. Model 1 using KNN had a recall of 0.9444 and only 2 false negatives. Model 2 using Logistic Regression also had a recall of 0.9444 and only 2 false negatives. Model 1 using KNN had a slightly better overall accuracy of 0.978 than Model 2 using Logistic Regression which had an accuracy of 0.967. Model 1 using KNN was also perfect when predicting a malignant outcome, which is the precision. Model 2 using Logistic Regression had a precision of 0.9714.

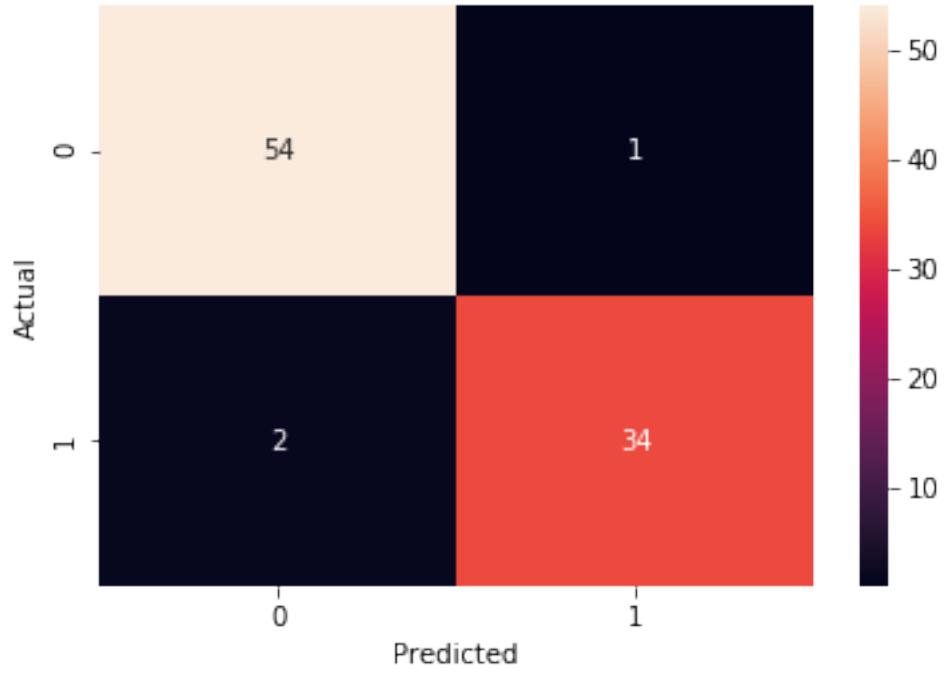
```
In [119]: ### Model 1 Knn
          plot_met(y_val, y_pred_all)
          scores(y_val, y_pred_all)

             precision    recall  f1-score   support
```

```
        0.0       0.96      1.00      0.98        55
        1.0       1.00      0.94      0.97        36

avg / total       0.98      0.98      0.98        91

Accuracy: 0.978021978021978
Accuracy: 0.978
Precision: 1.0
Recall: 0.9444
F1 Score: 0.9714
AUC: 0.9722
```




```
In [120]: #### Model 2
          plot_met(y_val, pred_cor)
          scores(y_val, pred_cor)

             precision    recall  f1-score   support

        0.0       0.96      0.98      0.97        55
        1.0       0.97      0.94      0.96        36

avg / total       0.97      0.97      0.97        91

Accuracy: 0.967032967032967
```

```
Accuracy: 0.967
Precision: 0.9714
Recall: 0.9444
F1 Score: 0.9577
AUC: 0.9631
```



I used 5 fold cross validation to see which of the two best models had a better f1 score when using cross validation. I took the average f1 score from the 5 folds. Based on this information a model was selected to be tested on the test data set.

```
In [121]: from sklearn.model_selection import cross_val_score

In [122]: new_X_train = X_train.append(X_val)
          new_y_train = y_train.append(y_val)

In [123]: model1 = KNeighborsClassifier(n_neighbors=3)

In [124]: model2 = LogisticRegression(penalty="l1")

In [125]: scoring = ['precision', 'recall', 'f1']
          print("Model 1: ")
          print(cross_val_score(model1, new_X_train, new_y_train, scoring="f1", cv = 5))
          recall = cross_val_score(model1, new_X_train, new_y_train, scoring="f1", cv = 5).mea
          print("Recall of Model 1 is: " , recall)
```

```
Model 1:
[0.95652174 0.95238095 0.90625    0.96969697 0.96875   ]
Recall of Model 1 is:  95.07199322416714
```

```
In [126]: print("Model 2: ")
          print(cross_val_score(model2, new_X_train[cor_features], new_y_train, scoring='f1', c
          recall = cross_val_score(model2, new_X_train[cor_features], new_y_train, scoring='f1
          print("Recall of Model 2 is: " , recall)
```

```
Model 2:
[0.93939394 0.91803279 0.9        0.97058824 0.95384615]
Recall of Model 2 is:  93.63722230838914
```

1.0.4 The Best Model

Model 1 using KNN was chosen because it had a better average f1 score than Model 2 using Logistic Regression. It also was the best model when trained with the training set and tested with the validation set. Model 1 using KNN had the best precision, recall, f1 score,auc, and accuracy on the validation set.

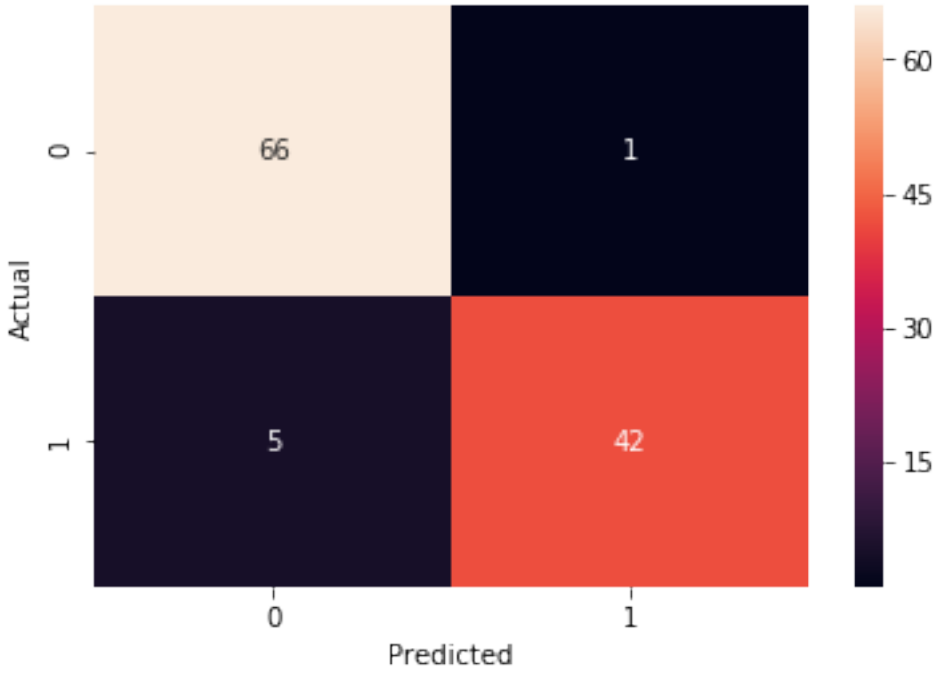
```
In [127]: ## Test Data
          model1_test = KNeighborsClassifier(n_neighbors=3)
          model1_test.fit(new_X_train, new_y_train)
```

```
Out[127]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                     weights='uniform')
```

```
In [128]: y_pred = model1_test.predict(X_test)
```

```
In [129]: plot_met(y_test, y_pred)
```

```
             precision    recall  f1-score   support

        0.0       0.93      0.99      0.96        67
        1.0       0.98      0.89      0.93        47

avg / total       0.95      0.95      0.95       114

Accuracy: 0.9473684210526315
```




```
In [130]: scores(y_test, y_pred)

Accuracy: 0.9474
Precision: 0.9767
Recall: 0.8936
F1 Score: 0.9333
AUC: 0.9393
```