[Jupyter Notebook](http://localhost:8888/tree?token=16fbac7fb5e37874d0c85da82341a82745bc2ed64f4f600f)

Data Science Capstone Healthcare-checkpoint(autosaved)Logout

Python 3

Trusted

* [File](http://localhost:8888/notebooks/Desktop/Project%202/Data%20Science%20Capstone%20Healthcare-checkpoint.ipynb)
* [Edit](http://localhost:8888/notebooks/Desktop/Project%202/Data%20Science%20Capstone%20Healthcare-checkpoint.ipynb)
* [View](http://localhost:8888/notebooks/Desktop/Project%202/Data%20Science%20Capstone%20Healthcare-checkpoint.ipynb)
* [Insert](http://localhost:8888/notebooks/Desktop/Project%202/Data%20Science%20Capstone%20Healthcare-checkpoint.ipynb)
* [Cell](http://localhost:8888/notebooks/Desktop/Project%202/Data%20Science%20Capstone%20Healthcare-checkpoint.ipynb)
* [Kernel](http://localhost:8888/notebooks/Desktop/Project%202/Data%20Science%20Capstone%20Healthcare-checkpoint.ipynb)
* [Widgets](http://localhost:8888/notebooks/Desktop/Project%202/Data%20Science%20Capstone%20Healthcare-checkpoint.ipynb)
* [Help](http://localhost:8888/notebooks/Desktop/Project%202/Data%20Science%20Capstone%20Healthcare-checkpoint.ipynb)

Run



In [1]:



**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**%**matplotlib inline

In [2]:



data **=**pd.read\_csv('health care diabetes.csv')

data.head()

​

Out[2]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

In [3]:



data.isnull().any()

​

Out[3]:

Pregnancies False

Glucose False

BloodPressure False

SkinThickness False

Insulin False

BMI False

DiabetesPedigreeFunction False

Age False

Outcome False

dtype: bool

In [4]:



data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 768 entries, 0 to 767

Data columns (total 9 columns):

Pregnancies 768 non-null int64

Glucose 768 non-null int64

BloodPressure 768 non-null int64

SkinThickness 768 non-null int64

Insulin 768 non-null int64

BMI 768 non-null float64

DiabetesPedigreeFunction 768 non-null float64

Age 768 non-null int64

Outcome 768 non-null int64

dtypes: float64(2), int64(7)

memory usage: 54.1 KB

In [5]:



Positive **=** data[data['Outcome']**==**1]

Positive.head(5)

​

Out[5]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 6 | 3 | 78 | 50 | 32 | 88 | 31.0 | 0.248 | 26 | 1 |
| 8 | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |

In [6]:



data['Glucose'].value\_counts().head(5)

Out[6]:

100 17

99 17

129 14

125 14

111 14

Name: Glucose, dtype: int64

In [7]:



plt.hist(data['Glucose'])

Out[7]:

(array([ 5., 0., 4., 32., 156., 211., 163., 95., 56., 46.]),

array([ 0. , 19.9, 39.8, 59.7, 79.6, 99.5, 119.4, 139.3, 159.2,

179.1, 199. ]),

<a list of 10 Patch objects>)

In [8]:



data['BloodPressure'].value\_counts().head(5)

​

Out[8]:

70 57

74 52

68 45

78 45

72 44

Name: BloodPressure, dtype: int64

In [9]:



plt.hist(data['BloodPressure'])

Out[9]:

(array([ 35., 1., 2., 13., 107., 261., 243., 87., 14., 5.]),

array([ 0. , 12.2, 24.4, 36.6, 48.8, 61. , 73.2, 85.4, 97.6,

109.8, 122. ]),

<a list of 10 Patch objects>)

In [10]:



data['SkinThickness'].value\_counts().head(5)

Out[10]:

0 227

32 31

30 27

27 23

23 22

Name: SkinThickness, dtype: int64

In [11]:



plt.hist(data['SkinThickness'])

Out[11]:

(array([231., 107., 165., 175., 78., 9., 2., 0., 0., 1.]),

array([ 0. , 9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99. ]),

<a list of 10 Patch objects>)

In [12]:



data['Insulin'].value\_counts().head(5)

Out[12]:

0 374

105 11

140 9

130 9

120 8

Name: Insulin, dtype: int64

In [13]:



plt.hist(data['Insulin'])

Out[13]:

(array([487., 155., 70., 30., 8., 9., 5., 1., 2., 1.]),

array([ 0. , 84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,

761.4, 846. ]),

<a list of 10 Patch objects>)

In [14]:



data['BMI'].value\_counts().head(5)

Out[14]:

32.0 13

31.6 12

31.2 12

0.0 11

33.3 10

Name: BMI, dtype: int64

In [15]:



plt.hist(data['BMI'])

Out[15]:

(array([ 11., 0., 15., 156., 268., 224., 78., 12., 3., 1.]),

array([ 0. , 6.71, 13.42, 20.13, 26.84, 33.55, 40.26, 46.97, 53.68,

60.39, 67.1 ]),

<a list of 10 Patch objects>)

In [16]:



data.describe().transpose()

Out[16]:

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pregnancies | 768.0 | 3.845052 | 3.369578 | 0.000 | 1.00000 | 3.0000 | 6.00000 | 17.00 |
| Glucose | 768.0 | 120.894531 | 31.972618 | 0.000 | 99.00000 | 117.0000 | 140.25000 | 199.00 |
| BloodPressure | 768.0 | 69.105469 | 19.355807 | 0.000 | 62.00000 | 72.0000 | 80.00000 | 122.00 |
| SkinThickness | 768.0 | 20.536458 | 15.952218 | 0.000 | 0.00000 | 23.0000 | 32.00000 | 99.00 |
| Insulin | 768.0 | 79.799479 | 115.244002 | 0.000 | 0.00000 | 30.5000 | 127.25000 | 846.00 |
| BMI | 768.0 | 31.992578 | 7.884160 | 0.000 | 27.30000 | 32.0000 | 36.60000 | 67.10 |
| DiabetesPedigreeFunction | 768.0 | 0.471876 | 0.331329 | 0.078 | 0.24375 | 0.3725 | 0.62625 | 2.42 |
| Age | 768.0 | 33.240885 | 11.760232 | 21.000 | 24.00000 | 29.0000 | 41.00000 | 81.00 |
| Outcome | 768.0 | 0.348958 | 0.476951 | 0.000 | 0.00000 | 0.0000 | 1.00000 | 1.00 |

**Data Exploration**

In [17]:



plt.hist(Positive['BMI'],histtype**=**'stepfilled',bins**=**20)

Out[17]:

(array([ 2., 0., 0., 0., 0., 0., 3., 13., 38., 61., 61., 36., 27.,

14., 7., 3., 1., 1., 0., 1.]),

array([ 0. , 3.355, 6.71 , 10.065, 13.42 , 16.775, 20.13 , 23.485,

26.84 , 30.195, 33.55 , 36.905, 40.26 , 43.615, 46.97 , 50.325,

53.68 , 57.035, 60.39 , 63.745, 67.1 ]),

<a list of 1 Patch objects>)

In [18]:



Positive['BMI'].value\_counts().head(7)

​

Out[18]:

32.9 8

31.6 7

33.3 6

30.5 5

32.0 5

31.2 5

32.4 4

Name: BMI, dtype: int64

In [19]:



plt.hist(Positive['Glucose'],histtype**=**'stepfilled',bins**=**20)

Out[19]:

(array([ 2., 0., 0., 0., 0., 0., 0., 1., 4., 9., 28., 26., 36.,

27., 29., 22., 24., 21., 25., 14.]),

array([ 0. , 9.95, 19.9 , 29.85, 39.8 , 49.75, 59.7 , 69.65,

79.6 , 89.55, 99.5 , 109.45, 119.4 , 129.35, 139.3 , 149.25,

159.2 , 169.15, 179.1 , 189.05, 199. ]),

<a list of 1 Patch objects>)

In [20]:



Positive['Glucose'].value\_counts().head(5)

Out[20]:

125 7

158 6

128 6

115 6

129 6

Name: Glucose, dtype: int64

In [21]:



plt.hist(Positive['BloodPressure'],histtype**=**'stepfilled',bins**=**20)

​

Out[21]:

(array([16., 0., 0., 0., 0., 1., 0., 1., 6., 6., 19., 37., 56.,

36., 41., 31., 7., 4., 4., 3.]),

array([ 0. , 5.7, 11.4, 17.1, 22.8, 28.5, 34.2, 39.9, 45.6,

51.3, 57. , 62.7, 68.4, 74.1, 79.8, 85.5, 91.2, 96.9,

102.6, 108.3, 114. ]),

<a list of 1 Patch objects>)

In [22]:



Positive['BloodPressure'].value\_counts().head(7)

Out[22]:

70 23

76 18

78 17

74 17

72 16

0 16

82 13

Name: BloodPressure, dtype: int64

In [23]:



plt.hist(Positive['SkinThickness'],histtype**=**'stepfilled',bins**=**20)

Out[23]:

(array([88., 1., 4., 10., 18., 30., 41., 34., 23., 15., 1., 1., 1.,

0., 0., 0., 0., 0., 0., 1.]),

array([ 0. , 4.95, 9.9 , 14.85, 19.8 , 24.75, 29.7 , 34.65, 39.6 ,

44.55, 49.5 , 54.45, 59.4 , 64.35, 69.3 , 74.25, 79.2 , 84.15,

89.1 , 94.05, 99. ]),

<a list of 1 Patch objects>)

In [24]:



plt.hist(Positive['Insulin'],histtype**=**'stepfilled',bins**=**20)

Out[24]:

(array([141., 6., 23., 33., 24., 12., 7., 7., 2., 1., 1.,

5., 3., 1., 1., 0., 0., 0., 0., 1.]),

array([ 0. , 42.3, 84.6, 126.9, 169.2, 211.5, 253.8, 296.1, 338.4,

380.7, 423. , 465.3, 507.6, 549.9, 592.2, 634.5, 676.8, 719.1,

761.4, 803.7, 846. ]),

<a list of 1 Patch objects>)

In [25]:



Positive['Insulin'].value\_counts().head(7)

Out[25]:

0 138

130 6

180 4

156 3

175 3

194 2

125 2

Name: Insulin, dtype: int64

**Scatter plot**

In [26]:



BloodPressure **=** Positive['BloodPressure']

Glucose **=** Positive['Glucose']

SkinThickness **=** Positive['SkinThickness']

Insulin **=** Positive['Insulin']

BMI **=** Positive['BMI']

In [27]:



plt.scatter(BloodPressure, Glucose, color**=**['b'])

plt.xlabel('BloodPressure')

plt.ylabel('Glucose')

plt.title('BloodPressure & Glucose')

plt.show()

In [28]:



g **=**sns.scatterplot(x**=** "Glucose" ,y**=** "BloodPressure",

hue**=**"Outcome",

data**=**data);

In [29]:



B **=**sns.scatterplot(x**=** "BMI" ,y**=** "Insulin",

hue**=**"Outcome",

data**=**data);

In [30]:



S **=**sns.scatterplot(x**=** "SkinThickness" ,y**=** "Insulin",

hue**=**"Outcome",

data**=**data);

In [31]:



*### correlation matrix*

data.corr()

Out[31]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pregnancies | 1.000000 | 0.129459 | 0.141282 | -0.081672 | -0.073535 | 0.017683 | -0.033523 | 0.544341 | 0.221898 |
| Glucose | 0.129459 | 1.000000 | 0.152590 | 0.057328 | 0.331357 | 0.221071 | 0.137337 | 0.263514 | 0.466581 |
| BloodPressure | 0.141282 | 0.152590 | 1.000000 | 0.207371 | 0.088933 | 0.281805 | 0.041265 | 0.239528 | 0.065068 |
| SkinThickness | -0.081672 | 0.057328 | 0.207371 | 1.000000 | 0.436783 | 0.392573 | 0.183928 | -0.113970 | 0.074752 |
| Insulin | -0.073535 | 0.331357 | 0.088933 | 0.436783 | 1.000000 | 0.197859 | 0.185071 | -0.042163 | 0.130548 |
| BMI | 0.017683 | 0.221071 | 0.281805 | 0.392573 | 0.197859 | 1.000000 | 0.140647 | 0.036242 | 0.292695 |
| DiabetesPedigreeFunction | -0.033523 | 0.137337 | 0.041265 | 0.183928 | 0.185071 | 0.140647 | 1.000000 | 0.033561 | 0.173844 |
| Age | 0.544341 | 0.263514 | 0.239528 | -0.113970 | -0.042163 | 0.036242 | 0.033561 | 1.000000 | 0.238356 |
| Outcome | 0.221898 | 0.466581 | 0.065068 | 0.074752 | 0.130548 | 0.292695 | 0.173844 | 0.238356 | 1.000000 |

In [32]:



*### create correlation heat map*

sns.heatmap(data.corr())

Out[32]:

<matplotlib.axes.\_subplots.AxesSubplot at 0xdf5e908>

In [33]:



plt.subplots(figsize**=**(8,8))

sns.heatmap(data.corr(),annot**=True**,cmap**=**'viridis') *### gives correlation value*

Out[33]:

<matplotlib.axes.\_subplots.AxesSubplot at 0xe476508>

In [34]:



plt.subplots(figsize**=**(8,8))

sns.heatmap(data.corr(),annot**=True**) *### gives correlation value*

Out[34]:

<matplotlib.axes.\_subplots.AxesSubplot at 0xe7347c8>

**Logistic Regreation and model building**

In [35]:



data.head()

Out[35]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

In [36]:



features **=** data.iloc[:,[0,1,2,3,4,5,6,7]].values

label **=** data.iloc[:,8].values

In [69]:



features

Out[69]:

array([[ 6. , 148. , 72. , ..., 33.6 , 0.627, 50. ],

[ 1. , 85. , 66. , ..., 26.6 , 0.351, 31. ],

[ 8. , 183. , 64. , ..., 23.3 , 0.672, 32. ],

...,

[ 5. , 121. , 72. , ..., 26.2 , 0.245, 30. ],

[ 1. , 126. , 60. , ..., 30.1 , 0.349, 47. ],

[ 1. , 93. , 70. , ..., 30.4 , 0.315, 23. ]])

In [37]:



**from** sklearn.model\_selection **import** train\_test\_split

X\_train,X\_test,y\_train,y\_test **=**train\_test\_split(features,label,test\_size **=**0.2,random\_state**=**10)

In [39]:



*###Create model*

**from** sklearn.linear\_model **import** LogisticRegression

model **=** LogisticRegression()

model.fit(X\_train,y\_train)

​

​

​

C:\Users\a1\Anaconda3\lib\site-packages\sklearn\linear\_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.

FutureWarning)

Out[39]:

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='warn', n\_jobs=None, penalty='l2',

random\_state=None, solver='warn', tol=0.0001, verbose=0,

warm\_start=False)

In [40]:



print(model.score(X\_train,y\_train))

print(model.score(X\_test,y\_test))

0.7833876221498371

0.7337662337662337

In [41]:



**from** sklearn.metrics **import** confusion\_matrix

cm **=** confusion\_matrix(label,model.predict(features))

cm

Out[41]:

array([[452, 48],

[126, 142]], dtype=int64)

In [42]:



**from** sklearn.metrics **import** classification\_report

clf **=**classification\_report(label,model.predict(features))

print(clf)

​

precision recall f1-score support

0 0.78 0.90 0.84 500

1 0.75 0.53 0.62 268

accuracy 0.77 768

macro avg 0.76 0.72 0.73 768

weighted avg 0.77 0.77 0.76 768

In [43]:



**from** sklearn.metrics **import** roc\_curve

**from** sklearn.metrics **import** roc\_auc\_score

​

*# predict probabilities*

probs **=** model.predict\_proba(features)

*# keep probabilities for the positive outcome only*

probs **=** probs[:, 1]

*# calculate AUC*

auc **=** roc\_auc\_score(label, probs)

print('AUC: %.3f' **%** auc)

*# calculate roc curve*

fpr, tpr, thresholds **=** roc\_curve(label, probs)

*# plot no skill*

plt.plot([0, 1], [0, 1], linestyle**=**'--')

*# plot the roc curve for the model*

plt.plot(fpr, tpr, marker**=**'.')

AUC: 0.834

Out[43]:

[<matplotlib.lines.Line2D at 0x10b02548>]

**Applying Decission Tree Classifier**

In [44]:



**from** sklearn.tree **import** DecisionTreeClassifier

model3 **=**DecisionTreeClassifier(max\_depth**=**5)

model3.fit(X\_train,y\_train)

​

​

Out[44]:

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=5,

max\_features=None, 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, presort=False,

random\_state=None, splitter='best')

In [45]:



model3.score(X\_train,y\_train)

Out[45]:

0.8289902280130294

In [46]:



model3.score(X\_test,y\_test)

Out[46]:

0.7662337662337663

**Applying Random Forest**

In [56]:



**from** sklearn.ensemble **import** RandomForestClassifier

model4 **=** RandomForestClassifier(n\_estimators**=**11)

model4.fit(X\_train,y\_train)

Out[56]:

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=11,

n\_jobs=None, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False)

In [57]:



model4.score(X\_train,y\_train)

Out[57]:

0.99185667752443

In [58]:



model4.score(X\_test,y\_test)

Out[58]:

0.7207792207792207

**Support Vector Classifier**

In [59]:



**from** sklearn.svm **import** SVC

model5 **=** SVC(kernel**=**'rbf',gamma**=**'auto')

model5.fit(X\_train,y\_train)

Out[59]:

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

In [60]:



model5.score(X\_train,y\_train)

Out[60]:

1.0

In [61]:



model5.score(X\_test,y\_test)

Out[61]:

0.6168831168831169

**Applying K-NN**

In [63]:



**from** sklearn.neighbors **import** KNeighborsClassifier

model2**=**KNeighborsClassifier(n\_neighbors**=**7,metric**=**'minkowski',

p **=** 2)

model2.fit(X\_train,y\_train)

Out[63]:

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=None, n\_neighbors=7, p=2,

weights='uniform')

In [65]:



model2.score(X\_train,y\_train)

Out[65]:

0.8045602605863192

In [66]:



model2.score(X\_test,y\_test)

Out[66]:

0.6948051948051948

In [71]:



*#Preparing ROC Curve (Receiver Operating Characteristics Curve)*

**from** sklearn.metrics **import** roc\_curve

**from** sklearn.metrics **import** roc\_auc\_score

​

*# predict probabilities*

probs **=** model2.predict\_proba(features)

*# keep probabilities for the positive outcome only*

probs **=** probs[:, 1]

*# calculate AUC*

auc **=** roc\_auc\_score(label, probs)

print('AUC: %.3f' **%** auc)

*# calculate roc curve*

fpr, tpr, thresholds **=** roc\_curve(label, probs)

print("True Positive Rate - {}, False Positive Rate - {} Thresholds - {}".format(tpr,fpr,thresholds))

*# plot no skill*

plt.plot([0, 1], [0, 1], linestyle**=**'--')

*# plot the roc curve for the model*

plt.plot(fpr, tpr, marker**=**'.')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

AUC: 0.836

True Positive Rate - [0. 0.06716418 0.23880597 0.44776119 0.60074627 0.75373134

0.88059701 0.98507463 1. ], False Positive Rate - [0. 0. 0.02 0.056 0.12 0.248 0.428 0.668 1. ] Thresholds - [2. 1. 0.85714286 0.71428571 0.57142857 0.42857143

0.28571429 0.14285714 0. ]

Out[71]:

Text(0, 0.5, 'True Positive Rate')

In [70]:



print('AUC: %.3f' **%** auc)

AUC: 0.834

In [72]:



*#Precision Recall Curve for Logistic Regression*

​

**from** sklearn.metrics **import** precision\_recall\_curve

**from** sklearn.metrics **import** f1\_score

**from** sklearn.metrics **import** auc

**from** sklearn.metrics **import** average\_precision\_score

*# predict probabilities*

probs **=** model.predict\_proba(features)

*# keep probabilities for the positive outcome only*

probs **=** probs[:, 1]

*# predict class values*

yhat **=** model.predict(features)

*# calculate precision-recall curve*

precision, recall, thresholds **=** precision\_recall\_curve(label, probs)

*# calculate F1 score*

f1 **=** f1\_score(label, yhat)

*# calculate precision-recall AUC*

auc **=** auc(recall, precision)

*# calculate average precision score*

ap **=** average\_precision\_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' **%** (f1, auc, ap))

*# plot no skill*

plt.plot([0, 1], [0.5, 0.5], linestyle**=**'--')

*# plot the precision-recall curve for the model*

plt.plot(recall, precision, marker**=**'.')

f1=0.620 auc=0.728 ap=0.728

Out[72]:

[<matplotlib.lines.Line2D at 0x1521b548>]

In [74]:



*#Precision Recall Curve for KNN*

​

**from** sklearn.metrics **import** precision\_recall\_curve

**from** sklearn.metrics **import** f1\_score

**from** sklearn.metrics **import** auc

**from** sklearn.metrics **import** average\_precision\_score

*# predict probabilities*

probs **=** model2.predict\_proba(features)

*# keep probabilities for the positive outcome only*

probs **=** probs[:, 1]

*# predict class values*

yhat **=** model2.predict(features)

*# calculate precision-recall curve*

precision, recall, thresholds **=** precision\_recall\_curve(label, probs)

*# calculate F1 score*

f1 **=** f1\_score(label, yhat)

*# calculate precision-recall AUC*

auc **=** auc(recall, precision)

*# calculate average precision score*

ap **=** average\_precision\_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' **%** (f1, auc, ap))

*# plot no skill*

plt.plot([0, 1], [0.5, 0.5], linestyle**=**'--')

*# plot the precision-recall curve for the model*

plt.plot(recall, precision, marker**=**'.')

f1=0.658 auc=0.752 ap=0.709

Out[74]:

[<matplotlib.lines.Line2D at 0xe671e48>]

In [75]:



*#Precision Recall Curve for Decission Tree Classifier*

​

**from** sklearn.metrics **import** precision\_recall\_curve

**from** sklearn.metrics **import** f1\_score

**from** sklearn.metrics **import** auc

**from** sklearn.metrics **import** average\_precision\_score

*# predict probabilities*

probs **=** model3.predict\_proba(features)

*# keep probabilities for the positive outcome only*

probs **=** probs[:, 1]

*# predict class values*

yhat **=** model3.predict(features)

*# calculate precision-recall curve*

precision, recall, thresholds **=** precision\_recall\_curve(label, probs)

*# calculate F1 score*

f1 **=** f1\_score(label, yhat)

*# calculate precision-recall AUC*

auc **=** auc(recall, precision)

*# calculate average precision score*

ap **=** average\_precision\_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' **%** (f1, auc, ap))

*# plot no skill*

plt.plot([0, 1], [0.5, 0.5], linestyle**=**'--')

*# plot the precision-recall curve for the model*

plt.plot(recall, precision, marker**=**'.')

f1=0.712 auc=0.812 ap=0.765

Out[75]:

[<matplotlib.lines.Line2D at 0x162dbf88>]

In [76]:



*#Precision Recall Curve for Random Forest*

​

**from** sklearn.metrics **import** precision\_recall\_curve

**from** sklearn.metrics **import** f1\_score

**from** sklearn.metrics **import** auc

**from** sklearn.metrics **import** average\_precision\_score

*# predict probabilities*

probs **=** model4.predict\_proba(features)

*# keep probabilities for the positive outcome only*

probs **=** probs[:, 1]

*# predict class values*

yhat **=** model4.predict(features)

*# calculate precision-recall curve*

precision, recall, thresholds **=** precision\_recall\_curve(label, probs)

*# calculate F1 score*

f1 **=** f1\_score(label, yhat)

*# calculate precision-recall AUC*

auc **=** auc(recall, precision)

*# calculate average precision score*

ap **=** average\_precision\_score(label, probs)

print('f1=%.3f auc=%.3f ap=%.3f' **%** (f1, auc, ap))

*# plot no skill*

plt.plot([0, 1], [0.5, 0.5], linestyle**=**'--')

*# plot the precision-recall curve for the model*

plt.plot(recall, precision, marker**=**'.')

f1=0.906 auc=0.964 ap=0.957

Out[76]:

[<matplotlib.lines.Line2D at 0x151f5c08>]

In [ ]:



​