# Linear regression

|  |  |
| --- | --- |
| Data Preparation | from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=101) |
| Creating the Model | from sklearn.linear\_model import LinearRegression  lm = LinearRegression() |
| Training the Model | lm.fit(X\_train,y\_train) |
| Evaluating the Model | print(lm.intercept\_)  coeff\_df = pd.DataFrame(lm.coef\_,X.columns,columns=['Coefficient'])  coeff\_df |
| Predictions | predictions = lm.predict(X\_test) |
| Metrics | from sklearn import metrics  print('MAE:', metrics.mean\_absolute\_error(y\_test, predictions))  print('MSE:', metrics.mean\_squared\_error(y\_test, predictions))  print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, predictions))) |

# Logistic Regression

|  |  |
| --- | --- |
| Data Preparation | sex = pd.get\_dummies(train['Sex'],drop\_first=True)  embark = pd.get\_dummies(train['Embarked'],drop\_first=True)  train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)  train = pd.concat([train,sex,embark],axis=1)  Fill blanks: sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(train.drop('Survived',axis=1),  train['Survived'], test\_size=0.30,  random\_state=101) |
| Creating the Model | from sklearn.linear\_model import LogisticRegression  logmodel = LogisticRegression() |
| Training the Model | logmodel.fit(X\_train,y\_train) |
| Evaluating the Model |  |
| Predictions | predictions = logmodel.predict(X\_test) |
| Metrics | from sklearn.metrics import classification\_report  print(classification\_report(y\_test,predictions)) |

# K Nearest Neighbors (Class prediction **– supervised**)

|  |  |
| --- | --- |
| Data Preparation | from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  scaler.fit(df.drop('TARGET CLASS',axis=1))  scaled\_features = scaler.transform(df.drop('TARGET CLASS',axis=1))  df\_feat = pd.DataFrame(scaled\_features,columns=df.columns[:-1])  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(scaled\_features,df['TARGET CLASS'],  test\_size=0.30) |
| Creating the Model | from sklearn.neighbors import KNeighborsClassifier  knn = KNeighborsClassifier(n\_neighbors=1) |
| Training the Model | knn.fit(X\_train,y\_train) |
| Evaluating the Model | #Getting accurate k values – the number of nearest neighbors – not the clusters  error\_rate = []  # Will take some time  for i in range(1,40):    knn = KNeighborsClassifier(n\_neighbors=i)  knn.fit(X\_train,y\_train)  pred\_i = knn.predict(X\_test)  error\_rate.append(np.mean(pred\_i != y\_test))  #Elbow method for K against the Error Rate  plt.figure(figsize=(10,6))  plt.plot(range(1,40),error\_rate,color='blue', linestyle='dashed', marker='o',  markerfacecolor='red', markersize=10)  plt.title('Error Rate vs. K Value')  plt.xlabel('K')  plt.ylabel('Error Rate') |
| Predictions | pred = knn.predict(X\_test) |
| Metrics | from sklearn.metrics import classification\_report,confusion\_matrix  print(confusion\_matrix(y\_test,pred))  print(classification\_report(y\_test,pred)) |

# Decision Trees

|  |  |
| --- | --- |
| Data Preparation | from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30) |
| Creating the Model | from sklearn.tree import DecisionTreeClassifier  dtree = DecisionTreeClassifier() |
| Training the Model | dtree.fit(X\_train,y\_train) |
| Evaluating the Model | Export Graph\_Viz |
| Predictions | predictions = dtree.predict(X\_test) |
| Metrics | from sklearn.metrics import classification\_report,confusion\_matrix  print(classification\_report(y\_test,predictions))  print(confusion\_matrix(y\_test,predictions)) |

# Random Forest

|  |  |
| --- | --- |
| Data Preparation | from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30) |
| Creating the Model | from sklearn.ensemble import RandomForestClassifier  rfc = RandomForestClassifier(n\_estimators=100) |
| Training the Model | rfc.fit(X\_train, y\_train) |
| Evaluating the Model |  |
| Predictions | rfc\_pred = rfc.predict(X\_test) |
| Metrics | print(confusion\_matrix(y\_test,rfc\_pred))  print(classification\_report(y\_test,rfc\_pred)) |

# Support Vector Machine

|  |  |
| --- | --- |
| Data Preparation | from sklearn.datasets import load\_breast\_cancer  cancer = load\_breast\_cancer()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_feat, np.ravel(df\_target), test\_size=0.30, random\_state=101) |
| Creating the Model | from sklearn.svm import SVC |
| Training the Model | model.fit(X\_train,y\_train) |
| Evaluating the Model | #If the above runs poorly, use the grid search given below  param\_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}  grid = GridSearchCV(SVC(),param\_grid,refit=True,verbose=3)  grid.fit(X\_train,y\_train)  grid.best\_params\_  grid.best\_estimator\_  grid\_predictions = grid.predict(X\_test) |
| Predictions | predictions = model.predict(X\_test) |
| Metrics | print(confusion\_matrix(y\_test,predictions))  print(classification\_report(y\_test,predictions)) |

# K Means clustering – Un supervised

|  |  |
| --- | --- |
| Data Preparation |  |
| Creating the Model | from sklearn.cluster import KMeans  kmeans = KMeans(n\_clusters=4) |
| Training the Model | kmeans.fit(data[0]) |
| Evaluating the Model | Visualization of the clusters  f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(10,6))  ax1.set\_title('K Means')  ax1.scatter(data[0][:,0],data[0][:,1],c=kmeans.labels\_,cmap='rainbow')  ax2.set\_title("Original")  ax2.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='rainbow') |
| Predictions | kmeans.labels\_  kmeans.cluster\_centers\_ |
| Metrics |  |

# Principal Component Analysis (PCA)

|  |  |
| --- | --- |
| Data Preparation | from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  scaler.fit(df)  scaled\_data = scaler.transform(df) |
| Creating the Model | from sklearn.decomposition import PCA  pca = PCA(n\_components=2) |
| Training the Model | x\_pca = pca.transform(scaled\_data) |
| Evaluating the Model | plt.figure(figsize=(8,6))  plt.scatter(x\_pca[:,0],x\_pca[:,1],c=cancer['target'],cmap='plasma')  plt.xlabel('First principal component')  plt.ylabel('Second Principal Component')  pca.components\_  df\_comp = pd.DataFrame(pca.components\_,columns=cancer['feature\_names'])  plt.figure(figsize=(12,6))  sns.heatmap(df\_comp,cmap='plasma',) |
| Predictions |  |
| Metrics |  |

# Neural Network

|  |  |
| --- | --- |
| Data Preparation | from sklearn.preprocessing import StandardScaler  ssc = StandardScaler()  ssc.fit(df.drop('Class',axis =1))  adf = ssc.transform(df.drop('Class',axis =1))  dfs = pd.DataFrame(adf , columns=df.drop('Class',axis =1).columns)  y = df['Class']  X = dfs.as\_matrix()  y = y.as\_matrix()  from sklearn.cross\_validation import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(dfs, y, test\_size=0.3, random\_state=42) |
| Creating the Model | import tensorflow.contrib.learn as learn  fnames = learn.infer\_real\_valued\_columns\_from\_input(X\_train)  dc = learn.DNNClassifier(n\_classes=2,hidden\_units=[10,20,10], feature\_columns=fnames) |
| Training the Model | dc.fit(X\_train,y\_train, steps = 200, batch\_size=20) |
| Evaluating the Model |  |
| Predictions | pred = dc.predict(X\_test) |
| Metrics | from sklearn.metrics import classification\_report,confusion\_matrix  #pred = list(pred)  print(confusion\_matrix(y\_test,pred))  print('\n')  print(classification\_report(y\_test,pred))  #pred |