: : : :	1t.ylabel('Cost Function') lt.ylabel('Cost Funct
	0.62751831 0.61607038 0.60501574 0.59434069 0.58403199 0.57407687 0.56446302 0.55517853 0.54621193 0.53755214 0.52918847 0.5211106 0.51330857 0.50577275 0.49849389 0.49146302 0.48467151 0.478111 0.47177345 0.4656511 0.45973644 0.45402224 0.44850151 0.44316752 0.43801375 0.43303392 0.42822199 0.42357209 0.41907858 0.41473602 0.41053915 0.40648289 0.40256234 0.39877279 0.39510965 0.39156854 0.3881452 0.38483553 0.38163557 0.37554151 0.37554965 0.37265643 0.36985843 0.36715232 0.3645349 0.362031 0.35955392 0.3571845 0.3461021 0.34078659 0.33902096 0.33731129 0.32675544 0.3244939 0.332489206 0.35267391 0.350952747 0.34845025 0.34643984 0.3444939 0.3324897 0.33099437 0.32099437 0.32953666 0.32812421 0.32675544 0.32542885 0.32414295 0.32289634 0.32168765 0.32051557 0.31937882 0.31827618 0.31141816 0.31054901 0.30970468 0.30970468 0.30980894 0.30970468
	0.30245103 0.30182899 0.30122321 0.30063312 0.30063819 0.29949788   0.2986517 0.29841915 0.29789976 0.29739308 0.29641612 0.299595   0.29548492 0.29508551 0.29459639 0.29178261 0.29416722 0.29374765   0.29337334 0.29293599 0.29035262 0.29018265 0.28967923 0.29178261 0.2914141   0.29105309 0.29069935 0.29035262 0.29001265 0.28967923 0.2895212   0.28903111 0.288716 0.28840657 0.28810264 0.28780402 0.28751053   0.28722199 0.28693824 0.28665911 0.26638445 0.2861141 0.28584793   0.28522199 0.28693824 0.2865910 0.28386677 0.28315815 0.289323   0.285285878 0.28532752 0.28507303 0.28482217 0.284582 0.28453087   0.28409019 0.28385269 0.28361825 0.28336677 0.28315815 0.2829323   0.28270913 0.28248854 0.28227046 0.28205479 0.28184145 0.28163038   0.2814215 0.28121472 0.28101 0.28080725 0.28060641 0.28040743   0.28021023 0.28001477 0.27982098 0.27962882 0.27943822 0.27924915   0.27906155 0.27788753 0.2786936 0.2783206 0.2773583 0.27708264   0.27706429 0.277778575 0.27760836 0.2773206 0.2772563 0.27708264   0.27706429 0.277778575 0.27760836 0.2773206 0.2772563 0.27708264   0.27090945 0.27673723 0.2760375 0.2763595 0.276355   0.27605751
	0.27588973
	0.26253184       0.26239379       0.26225588       0.26211812       0.26198049       0.261184301         0.26170566       0.2616845       0.26143137       0.26125757       0.2603955       0.2602036         0.26088441       0.260748       0.25993226       0.25996138       0.25956138       0.25939099         0.25925599       0.25912111       0.25885351       0.25885171       0.25877902       0.2577902         0.25944855       0.25813028       0.257245454       0.257212454       0.25712454       0.25614886       0.25651305       0.25644886       0.25654386       0.256544866       0.25654386       0.25556504       0.255539308         0.252526183       0.25240405       0.25433001       0.25433001       0.2533023       0.2543603       0.2543603         0.2536299       0.252378521       0.25252685       0.2533032       0.2533032       0.2534998       0.2530044         0.25214014       0.25278521       0.252526859       0.25252685       0.25253884       0.25252685       0.2523984       0.25249995
	0.25136964 0.2512416 0.25111367 0.25098585 0.25085813 0.25073052 0.250630302 0.25047562 0.25034834 0.25022115 0.2509408 0.24996711 0.24984025 0.24971349 0.24985864 0.2494603 0.24933386 0.24920753 0.2499813 0.24829056 0.24870325 0.2487744 0.24824065 0.24870325 0.24870325 0.24874498 0.247560 0.247546 0.24744989 0.247560 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2475047 0.2450404 0.2483045 0.2453045 0.2458040 0.2458
	0.2402632 0.2401444 0.24002569 0.23991077 0.23978856 0.23967014 0.23955183 0.2394336 0.23981548 0.23919746 0.23907953 0.2389617 0.23854396 0.23872633 0.23860879 0.23849134 0.23825675 0.2381396 0.2382224 0.23720585 0.2370857 0.2369738 0.2365729 0.23743871 0.23732224 0.23720585 0.23708957 0.23693785 0.23685729 0.23674129 0.2366523 0.23650958 0.23639387 0.23627825 0.23616273 0.236364731 0.23593198 0.23581674 0.2357016 0.23558655 0.2354716 0.236536675 0.23524199 0.23512732 0.23501275 0.23489827 0.23478388 0.2346696 0.2345554 0.23378866 0.23364521 0.2353186 0.2334186 0.2339584 0.2331928 0.2337886 0.2332965 0.2328537 0.2328537 0.23262839 0.23251588 0.23240346 0.23229113 0.232741 0.23262839 0.23317289 0.23161908 0.23150190 0.2315083 0.2380674 0.2312843 0.2317289 0.23106157 0.23085849 0.2308382 0.2300674 0.23061721 0.23080634 0.23093557 0.2308889 0.2301743 0.23006381
	0.2299534 0.22984308 0.22973286 0.2296272 0.22951267 0.22940272 0.22940272 0.2298535 0.22918308 0.2290334 0.22896338 0.22854348 0.2285632 0.228541718 0.22830812 0.22819916 0.22869028 0.22798149 0.2278278 0.22776419 0.2276557 0.22754724 0.2274389 0.22733064 0.22722248 0.22711441 0.22700642 0.22689852 0.22679071 0.22665753 0.225646781 0.22563036 0.22652999 0.22657536 0.225646781 0.22561063 0.22563093 0.22550388 0.22593141 0.2258244 0.22571747 0.22561063 0.22550388 0.22559721 0.22529063 0.22558474 0.22571747 0.22561063 0.2246519 0.22475905 0.22465299 0.226518414 0.22507774 0.22483519 0.2248519 0.22475905 0.2246529 0.22550388 0.2258378 0.22583781 0.22391303 0.23380767 0.2237039 0.2234921 0.22338708 0.2238215 0.2231731 0.22307255 0.2223652 0.2238215 0.2231731 0.22307255 0.2223652 0.222553755 0.22213356 0.2225503 0.2216545 0.2216545 0.2215144 0.221658 0.2215144 0.221658 0.2215144 0.221688 0.2255114 0.221688 0.2215114 0.2216869 0.2216859 0.2216836 0.2216836 0.2216836 0.2216836 0.2216836 0.2216836 0.2216836 0.2216838 0.2216836 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.2216838 0.22089528 0.2208928 0.2208928 0.2208938 0.2268659 0.2208836 0.22048123
	0.22037867 0.2202762 0.22017382 0.22007151 0.21986716 0.21986716 0.21976511 0.21966314 0.21965125 0.21945945 0.21945945 0.2193577 0.21895168 0.21885038 0.21874916 0.21864802 0.21854696 0.21844599 0.2183451 0.21824429 0.21814356 0.21864802 0.21734407 0.217274081 0.21734081 0.21734071 0.217274081 0.21734081 0.2164427 0.21574089 0.21844359 0.21844359 0.21844359 0.21844359 0.21844359 0.21844359 0.21844359 0.21844359 0.21844359 0.21844359 0.21844375 0.2154693 0.215464375 0.21644375 0.2154693 0.21644375 0.21644375 0.21644375 0.21644375 0.21644375 0.21644375 0.21644375 0.21644375 0.21565196 0.21555335 0.215654382 0.21476735 0.21646946 0.21457165 0.
	0.21148348 0.21138827 0.21129313 0.21119807 0.21109819 0.21008180 0.21091336 0.21081861 0.21025173 0.21015751 0.21062934 0.21062934 0.2095834 0.2095834 0.20978143 0.2096876 0.20959385 0.20995017 0.20940657 0.20931305 0.2092196 0.20912623 0.2093293 0.20983971 0.2084656 0.2087535 0.2092196 0.2085605 0.20856758 0.20847474 0.20838197 0.2082898 0.20819667 0.2081013 0.20801166 0.20791927 0.20782695 0.20773471 0.20764255 0.2081013 0.20801166 0.20791927 0.20782695 0.20773471 0.20764255 0.20745844 0.2073665 0.2067365 0.2067347 0.2078284 0.20791912 0.20699948 0.20690791 0.20681642 0.206725 0.20663365 0.20654238 0.20699948 0.20690791 0.2081042 0.2081042 0.2081043 0.2081046 0.2081043 0.2081046 0.2081044 0.2081046 0.208
	0.20268698
	0. 19553215 0. 19544987 0. 19536765 0. 1952855 0. 19520342 0. 1951214   0. 19503945 0. 19495756 0. 19487674 0. 19479399 0. 1947123 0. 19463088   0. 19454912 0. 19446763 0. 19438621 0. 19430485 0. 19422356 0. 19414233   0. 19406117 0. 19398008 0. 19381905 0. 19381808 0. 19373718 0. 19365635   0. 19357558 0. 19349487 0. 19341423 0. 19333366 0. 19325315 0. 19317271   0. 19309233 0. 19301201 0. 19293176 0. 19285158]
i tori	2- 1- 0 200 400 600 800 1000 Iterations
[2]: ir dr dr dr cr	Classification using Naive Bayes  mport pandas as pd  f = pd.read_csv("titanic.csv") f.head() f.info()  class 'pandas.core.frame.DataFrame'> angeIndex: 891 entries, 0 to 890 ata columns (total 12 columns): # Column Non-Null Count Dtype
	0 PassengerId 891 non-null int64 1 Survived 891 non-null int64 2 Pclass 891 non-null object 3 Name 891 non-null object 4 Sex 891 non-null object 5 Age 714 non-null float64 6 SibSp 891 non-null int64 7 Parch 891 non-null int64 8 Ticket 891 non-null object 9 Fare 891 non-null object 10 Cabin 204 non-null object types: float64(2), int64(5), object(5) emory usage: 83.7+ KB
d: 0 1 2 3 4	1
ta int[4]: 0 1 2 3 4	arget = df.Survived nputs.head()  Pclass Sex Age Fare  3 male 22.0 7.2500  1 female 38.0 71.2833  3 female 26.0 7.9250
dut[5]:	tummies         pd.get_dummies(inputs.Sex)           female         male           0         0         1           1         1         0           2         1         0           3         1         0           4         0         1           86         0         1
88 88 89 89	87       1       0         88       1       0         89       0       1
	nputs.drop(['Sex','male'],axis='columns',inplace=True) nputs.head(3)  Pclass Age Fare female 3 22.0 7.2500 0 1 38.0 71.2833 1
in n [8]: ii	
	1       35.0       53.1000       1         3       35.0       8.0500       0
[11]: fi	1 35.00000 53.100 1 3 35.00000 8.050 0 3 29.699118 8.4583 0  rom sklearn.model_selection import train_test_split _train, X_test, y_train, y_test = train_test_split(inputs, target, test_size=0.3)  rom sklearn.naive_bayes import GaussianNB odel = GaussianNB()
[12]:	GaussianNB()  _test[0:10]  _Polas
27 8 18 24 88 59	_test[0:10]
66 27 28 81 24 88 59 Na	42       1         65       0         76       0         73       0
[16]: y	cross checking  _predicted = model.predict(X_test)  rint('Accuracy on the training subset: {:.3f}'.format(model.score(X_train, y_train)))  rint('Accuracy on the test subset: {:.3f}'.format(model.score(X_test, y_test)))  ccuracy on the training subset: 0.788  ccuracy on the test subset: 0.754  rom sklearn.metrics import confusion_matrix onfusion_matrix(y_test,y_predicted)
[34]: ar	rray([[142, 38],
12 ## 12 ## pr pr	Creating Lasso model with alpha=0.1 asso = Lasso(alpha=0.1)  Fitting the model on the dataset asso.fit(X, y)  Printing the coefficients and intercept rint("Coefficients:", lasso.coef_) rint("Intercept:", lasso.intercept_)  oefficients: [3.16666667e-01 2.46716228e-17 0.00000000e+00] ntercept: 9.7333333333333333333333333333333333333
[35]: ir fi # X y # r: #	<pre>mport numpy as np rom sklearn.linear_model import Ridge  Creating a toy dataset = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) = np.array([10, 11, 12])  Creating Ridge model with alpha=0.1 idge = Ridge(alpha=0.1)  Fitting the model on the dataset idge.fit(X, y)</pre>
[36]: ##	Printing the coefficients and intercept rint("Coefficients:", ridge.coef_) rint("Intercept:", ridge.intercept_)  oefficients: [0.11090573 0.11090573 0.11090573] ntercept: 9.336414048059149  Classification using Logistic Regression  Loading the Data  mport pandas as pd mport numpy as np
ir ir fr fr t:	mport sklearn as sl mport matplotlib.pyplot as plt mport seaborn as sns  rom sklearn import metrics rom sklearn.linear_model import LogisticRegression rom sklearn.model_selection import train_test_split  itanic = pd.read_csv("Titanic.csv")  itanic.head()  Passengerld Survived Pclass Name Sex Age SibSp Parch Ticket Fare Cabin Embarked  1 0 3 Braund.Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S
1 2 3 4	2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th female 38.0 1 0 PC 17599 71.2833 C85 C 3 1 3 Heikkinen, Miss. Laina female 26.0 0 0 STON/O2. 3101282 7.9250 NaN S 4 1 1 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0 113803 53.1000 C123 S
р	Embarked_C         Embarked_Q         Embarked_S           0         0         1           1         0         0           0         1           0         1           0         0
propriet [21]:  0 1 2 3 4	itanic = titanic.join(ports) itanic.drop(['Embarked'], axis=1, inplace= <b>True</b> )
[22]: t: [23]: # t: [24]: # t: [24]: #	

4 0

Accuracy: 0.8097014925373134 Precision: 0.78

In [30]: #To evaluate the entire test set, we can use the metrics module from the scikit-learn package.
print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))
print("Precision:", metrics.precision\_score(y\_test, y\_pred))
print("Recall:", metrics.recall\_score(y\_test, y\_pred))

Name: Nehal ----- Roll No.: 2020332 ----- Subject: Machine Learning

Assignment 1

In [1]: def cost\_function(X, y, theta):
 m = len(y)

return J

m = len(y)

Linear Regression with Gradient descent

J = np.sum((X.dot(theta) - y) \*\* 2) / (2 \* m)

def gradient\_descent(X, y, theta, alpha, iterations):

J\_history = np.zeros(iterations)

Recall: 0.7289719626168224