Data Science Modeling Basics

Presented by the IBM Data Science Apprentice Team

Darrel, Denise, Jesse, Jessica & Samuel

Table of Contents

This Photo This Photo CC BY-SA CC BY-NC

Beginner Resources

Online references

codecademy

PythonForBeginners

edX

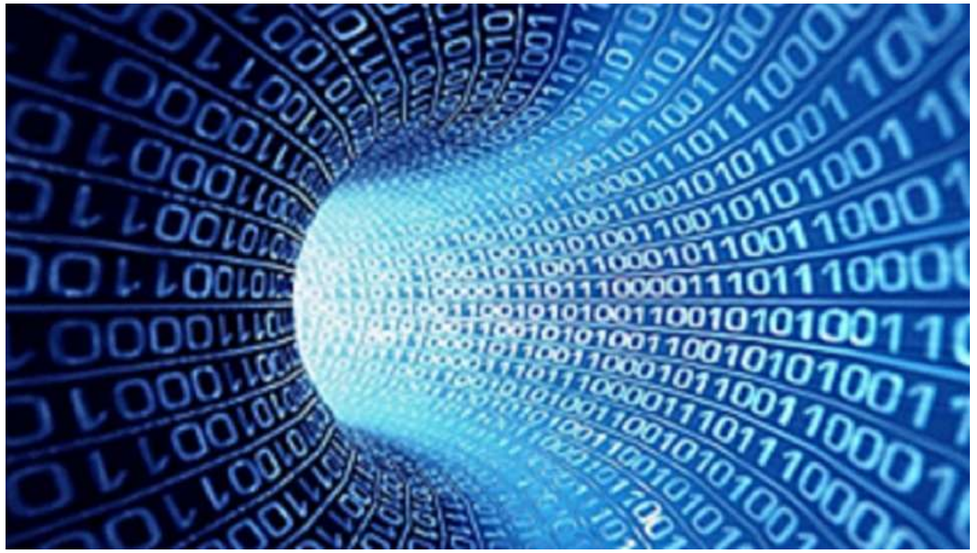
coursera

Example projects

kaggle

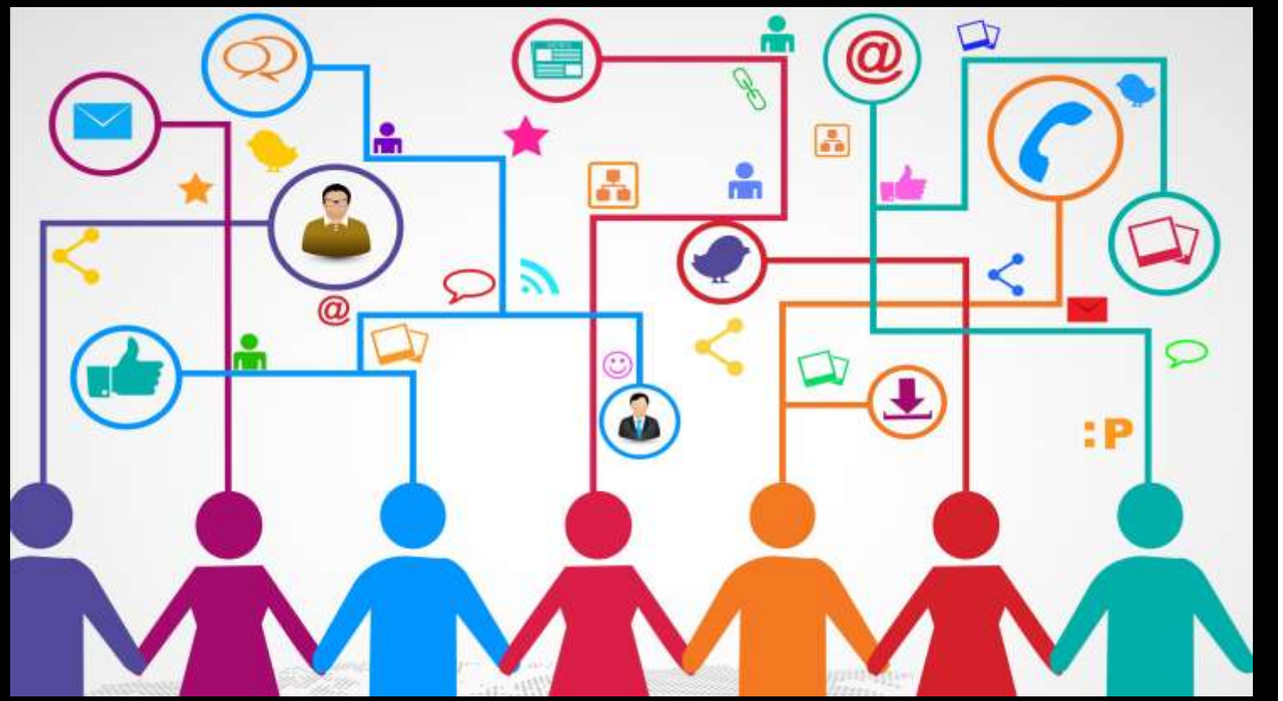
Your mentors!

slack



What is classification?

Classification: the process of predicting the *class* of given data points



Spam identification

Real vs. fake social media accounts

Image processing

Requires:

Features

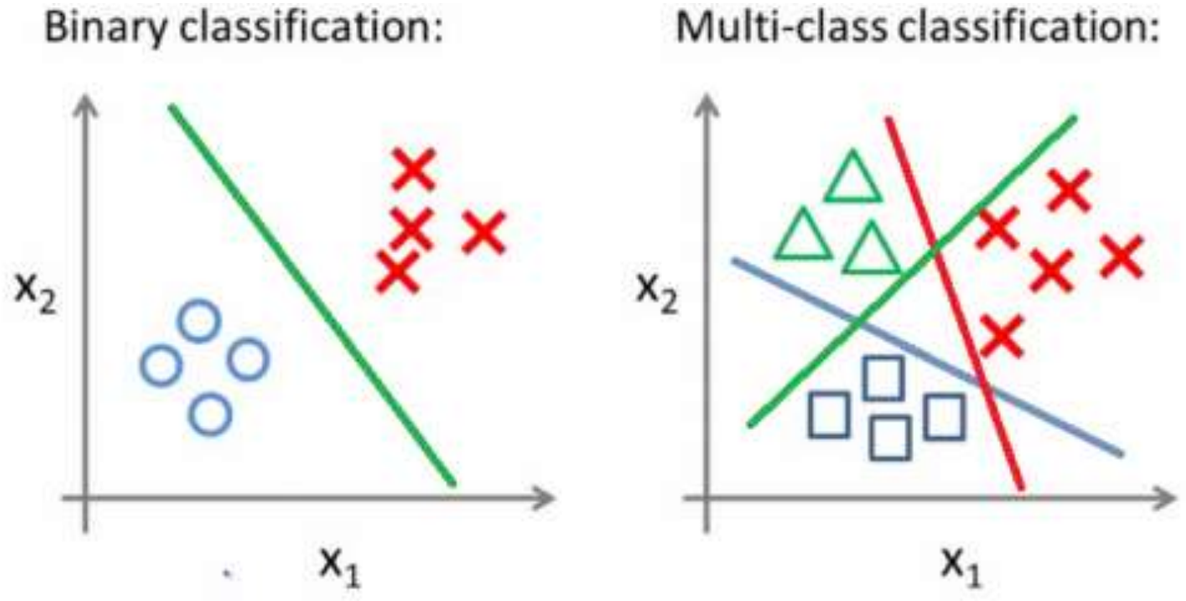
Targets

Algorithm

K-nearest-neighbor (KNN), random forest, decision tree, etc.

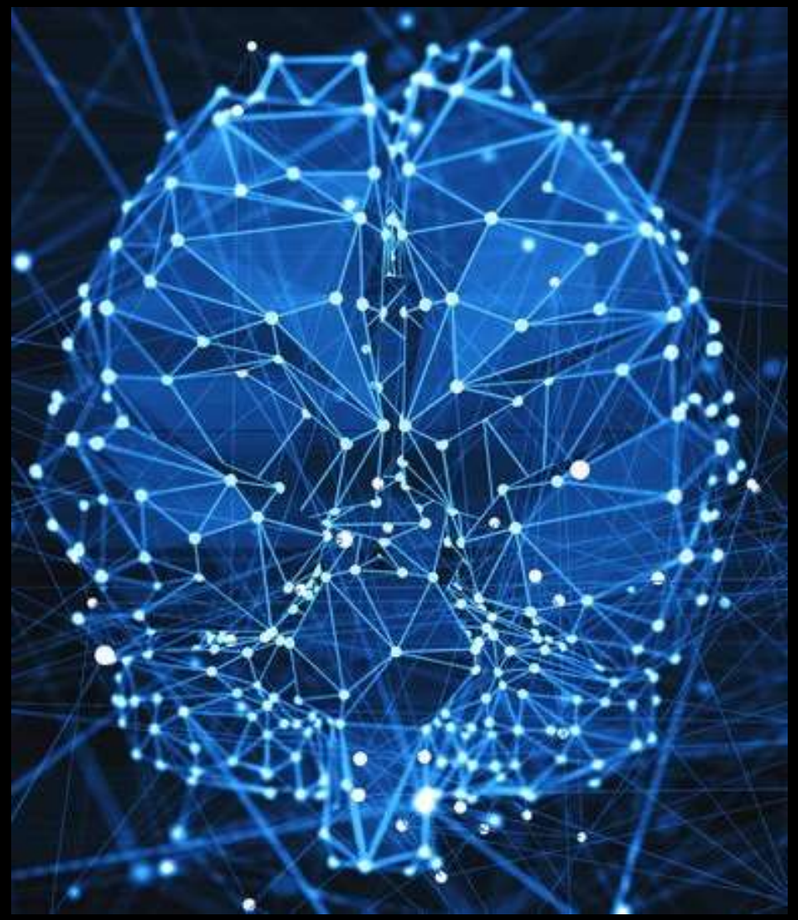
Bi-class (Binary) vs. Multi-class Classification

- Binary classification: classifying data points as either 'A' or 'B'
  - -Medical diagnoses: sick vs. healthy
- Multi-class classification: classifying data points as 'A', 'B', 'C', … , or 'Z'
  - -Image identification: what animal is this?



Source: https://medium.com/@b.terryjack/tips-and-tricks-for-multi-class-classification-c184ae1c8ffc

Classification Walkthrough



Overview:

Step 1: Exploring the Data

Step 2: Preparing the Data

Step 3: Model Implementation

Step 4: Evaluate the Model

Step 1: Exploring the Data

This step involves exploring the data through exploratory data visualizations and analyses to see what we have and what we need.

Step 2: Preparing the Data

In this section we would usually tidy our data to make sure it is clean in order to feed it into a model and have it properly run. For our purposes your data will come ready but this is a critical step in the classification process.

This involves:

- Looking for and replacing missing values.
- Removing erroneous features (i.e. features we don't think will have an effect on our outcome).
- Normalize features (transform skewed continuous features, normalizing numerical features, one-hot encode non-numeric features)

Once all categorical data has been transformed into numerical, and all numerical data has been normalized, we will split the data into training and test sets. 80% will be used for training and 20% for testing.

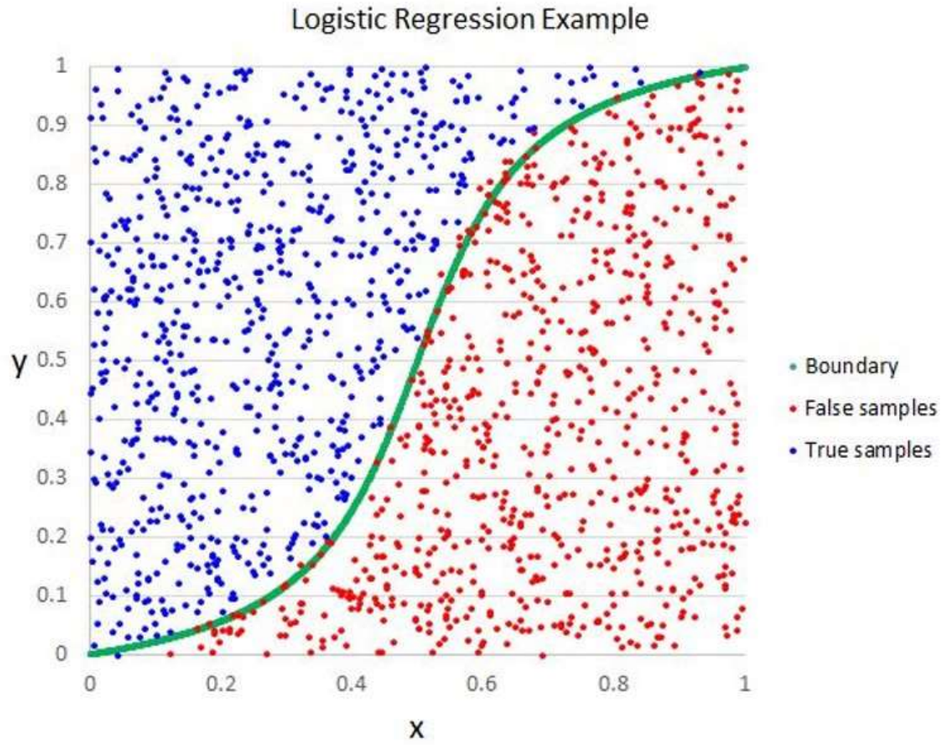
Step 3: Model Implementation

Our data is ready to be used in a model! There are many model architectures out there. Today we will focus on one:

Logistic Regression

Some info on Logistic Regression

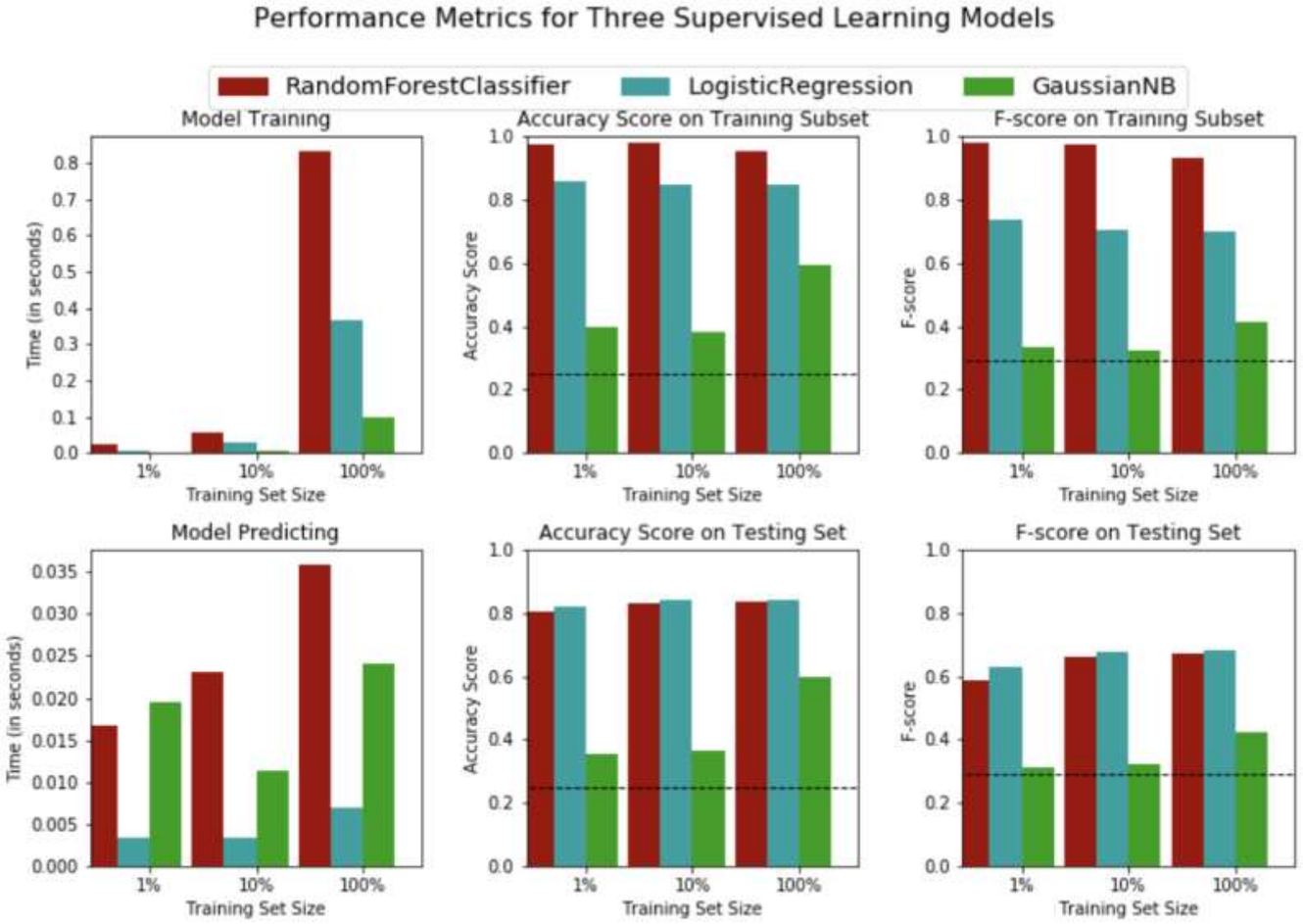
•Logistic Regression is a good model when we are looking for a binary outcome (yes or no, 0 or 1, normal or abnormal).



Source: https://www.datasciencecentral.com/profiles/blogs/why-logistic-regression-should-be-the-last-thing-you-learn-when-b

Step 4: Evaluate the Model

During this step we evaluate our model using performance metrics that are a part of the python libraries we use for our model.



Python Libraries

An open sourced package that provides high-performance easy to use data structures and analysis tools allowing you to format your data into a data frame where you can easily index, manipulate, rename, sort, and merge the data frame as you please. It also allows you to handle missing data, plot your data, and update, add, and delete columns.

NumPy is a general-purpose array processing package. NumPy is used to process arrays that store values of the same datatype. You will use NumPy to 1. perform basic array operations such as adding, multiplying, flatten, reshape, and index arrays 2. Work with DateTime or Linear Algebra. 3. Basic slicing and advanced indexing.

Matplotlib is a visualization library that enables you to plot 2D visualizations. It allows you to create line plots, scatter plots, area plots, bar charts and histograms, pie charts, stem plots, contour plots, quiver plots, and spectrograms.

Scikit Learn is a machine learning library for Python. It features numerous machine learning algorithms such as random forests, k-means clustering, cross-validation and more. If you're aiming to use supervised learning models like Naïve Bayes or grouping unlabeled data through KMeans, SciKit learn is the ideal library

Importing Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Pandas and Matplotlib Examples

[2]:
```
df = pd.read_csv('/kaggle/input/titanic/train.csv')
```

[40]:
```
df
```

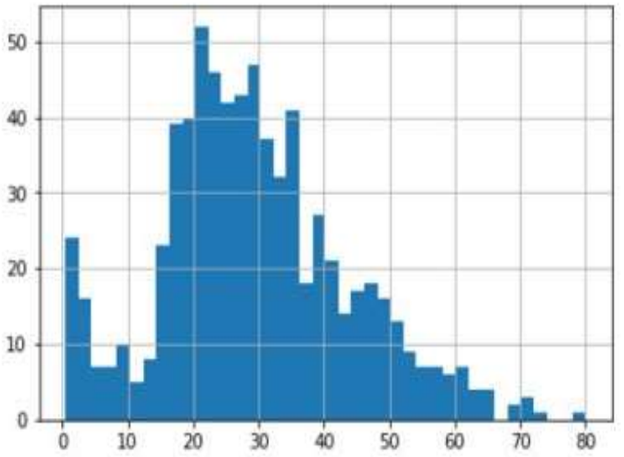
Out[40]:

	Survived	Pclass	Age	SibSp	Parch	Fare	male
0	0	3	22.0	1	0	7.2500	1
1	1	1	38.0	1	0	71.2833	0
2	1	3	26.0	0	0	7.9250	0
3	1	1	35.0	1	0	53.1000	0
4	0	3	35.0	0	0	8.0500	1
...	...	...	...	...	...	...	...
886	0	2	27.0	0	0	13.0000	1
887	1	1	19.0	0	0	30.0000	0
888	0	3	27.0	1	2	23.4500	0
889	1	1	26.0	0	0	30.0000	1
890	0	3	32.0	0	0	7.7500	1

891 rows × 7 columns

```
df['Age'].hist(bins = 40)
```

ut[13]
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd4a9170860>
```



Using Scikit-learn for modeling

Model Training

[42]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = .20, random_state = 0)
```

[43]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train,y_train)
```

Out[43]
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Exploratory Data Analysis

In statistics, **exploratory data analysis** (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

Basic EDA Code

```
df.head()
```

]:

	checkingstatus	loanduration	credithistory	loanpurpose	loanamount	existingsavings	employmentduration	installmentpercent	sex	othersonloan	currentresidenceduration	ownsproperty
0	less_0	9	prior_payments_delayed	furniture	2219	less_100	1_to_4	1	female	none	1	savings_insurance
1	less_0	35	credits_paid_to_date	car_used	389	100_to_500	4_to_7	3	female	none	2	car_other
2	0_to_200	19	credits_paid_to_date	repairs	2083	500_to_1000	less_1	3	female	none	2	real_estate
3	no_checking	36	prior_payments_delayed	appliances	5927	unknown	greater_7	4	male	co-applicant	3	savings_insurance
4	no_checking	28	outstanding_credit	appliances	4053	unknown	greater_7	5	male	none	3	unknown

```
df.tail()
```

	checkingstatus	loanduration	credithistory	loanpurpose	loanamount	existingsavings	employmentduration	installmentpercent	sex	othersonlo
4822	no_checking	0.333333	prior_payments_delayed	appliances	0.549711	500_to_1000	greater_7	0.6	male	guarant
139	no_checking	0.416667	prior_payments_delayed	car_used	0.562402	100_to_500	4_to_7	0.6	male	no
4752	no_checking	0.600000	outstanding_credit	repairs	0.632942	greater_1000	greater_7	1.0	male	guarant
3739	no_checking	0.600000	outstanding_credit	other	0.559076	unknown	greater_7	0.8	male	guarant
1878	no_checking	0.616667	outstanding_credit	appliances	0.608962	unknown	greater_7	1.0	male	guarant

```
df.columns
```

```
]: Index(['checkingstatus', 'loanduration', 'credithistory', 'loanpurpose',
        'loanamount', 'existingsavings', 'employmentduration',
        'installmentpercent', 'sex', 'othersonloan', 'currentresidenceduration',
        'ownsproperty', 'age', 'installmentplans', 'housing',
        'existingcreditscount', 'job', 'dependents', 'telephone',
        'foreignworker', 'risk'],
       dtype='object')
```

```
df.shape
```

```
]: (5000, 21)
```

More EDA Code

```
df.describe()
```

]:

	loanduration	loanamount	installmentpercent	currentresidenceduration	age	existingcreditscount	dependents
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	21.393000	3480.145000	2.982400	2.854200	35.932400	1.465800	1.164600
std	11.162843	2488.232783	1.127096	1.115702	10.648536	0.565415	0.370856
min	4.000000	250.000000	1.000000	1.000000	19.000000	1.000000	1.000000
25%	13.000000	1326.750000	2.000000	2.000000	28.000000	1.000000	1.000000
50%	21.000000	3238.500000	3.000000	3.000000	36.000000	1.000000	1.000000
75%	29.000000	5355.000000	4.000000	4.000000	44.000000	2.000000	1.000000
max	64.000000	11676.000000	6.000000	6.000000	74.000000	4.000000	2.000000

```
df.nunique()
```

```
checkingstatus              4
loanduration               59
credithistory               5
loanpurpose                11
loanamount               3354
existingsavings             5
employmentduration          5
installmentpercent          6
sex                         2
othersonloan                3
currentresidenceduration    6
ownsproperty                4
age                        53
installmentplans            3
housing                     3
existingcreditscount        4
job                         4
dependents                  2
telephone                   2
foreignworker               2
risk                        2
dtype: int64
```

```
df.dtypes
```

]:

```
checkingstatus              object
loanduration                 int64
credithistory               object
loanpurpose                 object
loanamount                   int64
existingsavings             object
employmentduration          object
installmentpercent           int64
sex                         object
othersonloan                object
currentresidenceduration     int64
ownsproperty                object
age                          int64
installmentplans            object
housing                     object
existingcreditscount         int64
job                         object
dependents                   int64
telephone                   object
foreignworker               object
risk                        object
dtype: object
```

Indexing and Referencing your data

What other things can we do with Pandas dataframes?

1. Queries
2. Finding specific values
3.Group by
4. Lists and appending to lists

	year	month	carrier	carrier_name	airport	airport_name	arr_flights	arr_del15	carrier_ct	weather_ct	nas_ct	security_ct	late_aircraft_ct	arr_
0	2016	1	DL	Delta Air Lines Inc.	MSP	Minneapolis, MN: Minneapolis-St Paul Internati...	4316.0	526.0	166.06	22.06	176.35	0.00	161.53	
1	2016	1	DL	Delta Air Lines Inc.	LAX	Los Angeles, CA: Los Angeles International	2623.0	577.0	118.37	19.98	290.48	0.00	148.17	
2	2016	1	DL	Delta Air Lines Inc.	BOS	Boston, MA: Logan International	1216.0	179.0	52.27	11.37	63.61	0.00	51.75	
3	2016	1	DL	Delta Air Lines Inc.	ATL	Atlanta, GA: Hartsfield-Jackson Atlanta Intern...	19115.0	2179.0	644.81	101.28	716.62	0.00	716.30	
4	2016	1	DL	Delta Air Lines Inc.	MCO	Orlando, FL: Orlando International	1554.0	247.0	70.55	15.13	97.58	0.00	63.74	
5	2016	1	DL	Delta Air Lines Inc.	PHL	Philadelphia, PA: Philadelphia International	532.0	46.0	19.32	2.51	14.46	0.00	9.70	

Queries

Find specific parts of your data by using df.query

```
In [45]: df.query('year > 2015')
```

Out[45]:

	year	month	carrier	carrier_name	airport	airport_name	arr_flights	arr_del15	carrier_ct	weather_ct	...	arr_cancelled	arr_diverted	arr_delay
0	2016	1	DL	Delta Air Lines Inc.	MSP	Minneapolis, MN: Minneapolis-St Paul Internati...	4316.0	526	166.06	22.06	...	33.0	6	35267
1	2016	1	DL	Delta Air Lines Inc.	LAX	Los Angeles, CA: Los Angeles International	2623.0	577	118.37	19.98	...	22.0	1	33960
2	2016	1	DL	Delta Air Lines Inc.	BOS	Boston, MA: Logan International	1216.0	179	52.27	11.37	...	17.0	2	11470
3	2016	1	DL	Delta Air Lines Inc.	ATL	Atlanta, GA: Hartsfield-Jackson Atlanta Intern...	19115.0	2179	644.81	101.28	...	221.0	10	153088
4	2016	1	DL	Delta Air Lines Inc.	MCO	Orlando, FL: Orlando International	1554.0	247	70.55	15.13	...	19.0	1	12447

Finding the max and min of a column

You can also find specific values of your df by using .max(), .min() or .loc()

```
: #find the airport with the most total delays
df[df['arr_delay']==df['arr_delay'].max()]
```

:

	year	month	carrier	carrier_name	airport	airport_name	arr_flights	arr_del15	carrier_ct	weather_ct	...	late_aircraft_ct	arr_cancelled	arr_diverted	arr
450	2017	4	DL	Delta Air Lines Inc.	ATL	Atlanta, GA: Hartsfield-Jackson Atlanta Intern...	20424.0	3370.0	1027.45	112.19	...	1321.37	1197.0	101	4

```
In [117]: df.loc[df['airport'] == 'BOS']
```

Out[117]:

	year	month	carrier	carrier_name	airport	airport_name	arr_flights	arr_del15	carrier_ct
2	2016	1	DL	Delta Air Lines Inc.	BOS	Boston, MA: Logan International	1216.0	179	52.27
49	2016	2	DL	Delta Air Lines Inc.	BOS	Boston, MA: Logan International	1208.0	243	49.24
78	2016	3	DL	Delta Air	BOS	Boston, MA: Logan	1242.0	202	62.13

Grouping

I would compare this Pandas feature to a pivot table, for example. You can group your data by one or more columns. This works best with categorical features for the first ”groupby” column.

```
In [31]: df.groupby('airport')['arr_delay_avg'].mean().head(10)

Out[31]: airport
         ATL     8.670568
         BOS    13.831421
         BWI     7.732192
         CLT     7.414518
         DCA     9.448472
         DEN     8.711689
         DFW     8.368311
         DTW     8.077908
         EWR    20.004214
         FLL     9.093329
```

In this cell, we are grouping the airports by the average time flights were delayed. The .head(10) shows only the top ten results. Thus, ATL airport has longest delays.

Lists

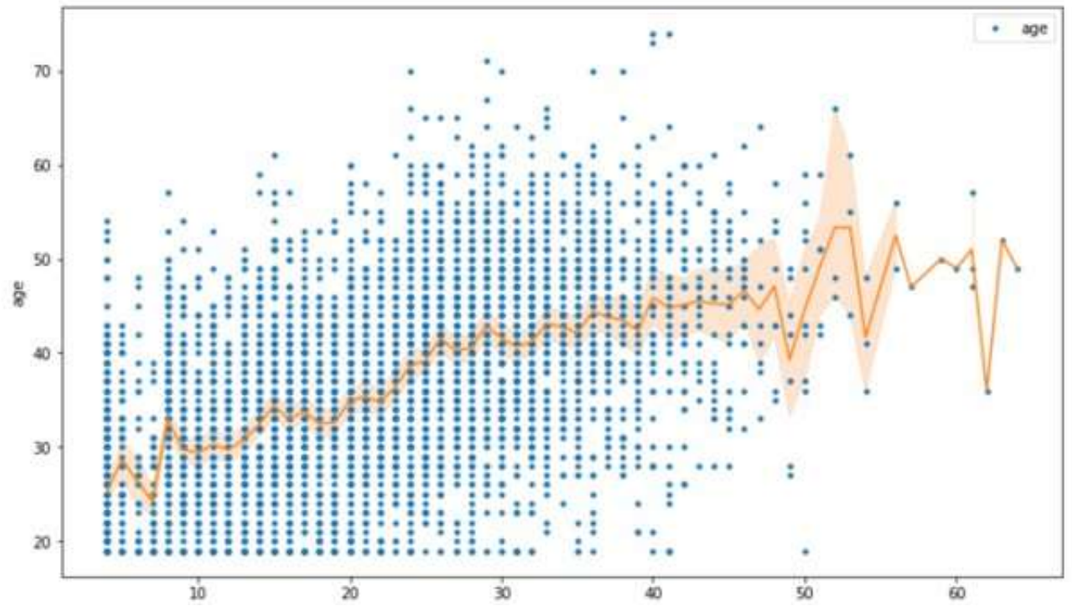
A pandas column, or Series, can also come from lists. You can make lists with this data and append to it.

```
In [61]: Months = ['January', 'February', 'March', 'April', 'May', 'June',
                   'July', 'August', 'September', 'October', 'November', 'December']
         #Create an empty list
         months_so_far = []
         for month in Months[:3]:
             months_so_far.append(month)

In [62]: months_so_far

Out[62]: ['January', 'February', 'March']
```

Basic data visualization for EDA

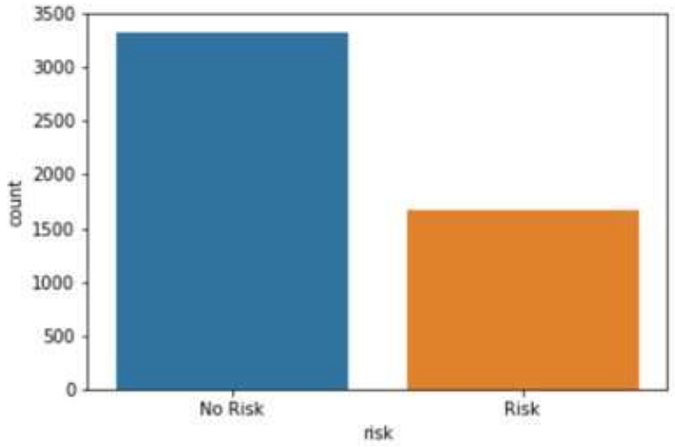
```
import seaborn as sns
from matplotlib import pyplot as plt
```

```
plt.figure(figsize=(12,7))
plt.plot(df.loanduration, df.age,'.', alpha = 1)
sns.lineplot(x=df.loanduration, y=df.age)
plt.show()
```



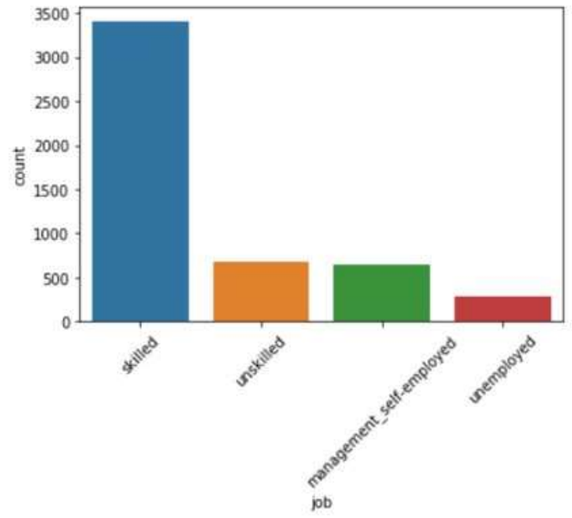

```
sns.countplot(df.risk)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11c8b6908>
```




```
sns.countplot(df.job)
plt.xticks(rotation=45)
```

```
(array([0, 1, 2, 3]), <a list of 4 Text xticklabel ob
```



Python notebook: Logistic Binary Regression

Step 1: Data Ingestion and Verification

```
In [22]: df = pd.read_csv(dataFile)
```

```
In [24]: df.describe()
```

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Source: https://www.kaggle.com/uciml/pima-indians-diabetes-database

Python notebook: Logistic Binary Regression

Step 2: Data Exploration

```
In [35]: sns.pairplot(data=df,hue='Outcome')
```

```
Out[35]: <seaborn.axisgrid.PairGrid at 0x7faa387a8198>
```

Python notebook: Logistic Binary Regression

Step 4: Data prep

```
In [43]: X = ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']
         y = ['Output']
```

```
In [44]: df2 = pd.DataFrame(data=df)
         df2.head()
```

Out[44]:

	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

Python notebook: Logistic Binary Regression

Step 5: Build the model

```
In [45]: X_train, X_test, y_train, y_test = train_test_split(df.drop('Outcome',axis=1),df['Outcome'],
                                                             test_size=0.30, random_state=101)

In [46]: from sklearn.linear_model import LogisticRegression

In [47]: logmodel = LogisticRegression()

In [48]: logmodel.fit(X_train,y_train)

Out[48]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, max_iter=100, multi_class='warn',
                            n_jobs=None, penalty='l2', random_state=None, solver='warn',
                            tol=0.0001, verbose=0, warm_start=False)
```

Python notebook: Logistic Binary Regression

Step 6: Evaluate the model

```
In [49]: predictions = logmodel.predict(X_test)

In [50]: from sklearn.metrics import classification_report, confusion_matrix

In [51]: print(classification_report(y_test,predictions))
```

```
                precision    recall  f1-score   support

             0       0.79      0.91      0.84       150
             1       0.76      0.56      0.64        81

     micro avg       0.78      0.78      0.78       231
     macro avg       0.78      0.73      0.74       231
  weighted avg       0.78      0.78      0.77       231
```

Thanks!

Contact Information:

Serena Bellesi: Serena.Bellesi@ibm.com

Denise Hernandez: Denise.Hernandez@ibm.com

Jessica Horvath: Jessica.Horvath@ibm.com

Samuel Martin: Samuel.Martin1@ibm.com

Darrel Moxam: Darrel.Moxam@ibm.com

Jesse Rhodes: Jesse.Rhodes@ibm.com