

Building a Machine Learning Model (inspired by Microsoft)

Exercise 1: Create an Azure notebook

Azure notebooks are contained in projects, whose primary purpose is to group related notebooks. In this exercise, you will create a new project and then create a notebook inside it.

1. Navigate to <https://notebooks.azure.com> in your browser and sign in using your Microsoft account. Click My Projects in the menu at the top of the page. Then click the + New Project button at the top of the "My Projects" page.
2. Create a new project named "ML Notebooks" or something similar. You may uncheck the "Public" box if you'd like, but making the project public allows the notebooks in it to be shared with others through links, social media, or e-mail. If you're unsure which to choose, you can easily change a project to public or private later on.

?

×

Create New Project

You can create a blank project where your notebooks and files will be placed

Project Name

Project ID ?

jeffpro/projects/

☒ Public

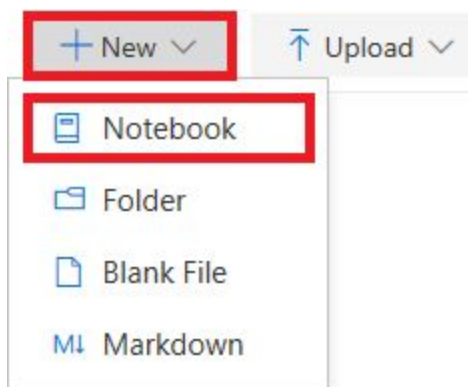
☒ Initialize this project with a README

Create

Cancel

Creating a project

3. Click + New and select Notebook from the menu to add a notebook to the project.



Adding a notebook to the project

4. Give the notebook a name such as "On-Time Flight Arrivals.ipynb", and select Python 3.6 as the language. This will create a notebook with a Python 3.6 kernel for executing Python code. One of the strengths of Azure notebooks is that you can use different languages by choosing different kernels.

Create New Notebook

Finish creating a new notebook by filling in the name and selecting the language.

Notebook Name

On-Time Flight Arrivals.ipynb

Select Language

☐ Python 2.7

☐ Python 3.5

☒ Python 3.6

☐ R

☐ F#

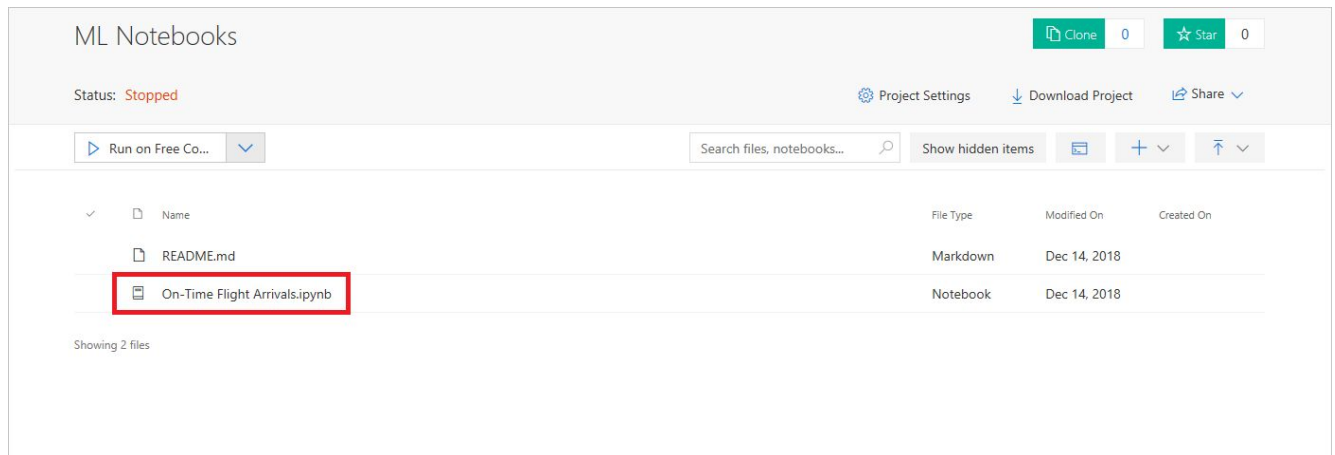
New Cancel

Creating a notebook

If you're curious, the .ipynb file-name extension stands for "IPython notebook." Jupyter

notebooks were originally known as IPython (Interactive Python) notebooks, and they only supported Python as a programming language. The name Jupyter is a combination of Julia, Python, and R — the core programming languages that Jupyter supports.

5. Click the notebook to open it for editing.



Opening the notebook

You can create additional projects and notebooks as you work with Azure Notebooks. You can create notebooks from scratch, or you can upload existing notebooks. In the next exercise, you will import a dataset and load it into the notebook you created.

Exercise 2: Import and load a dataset

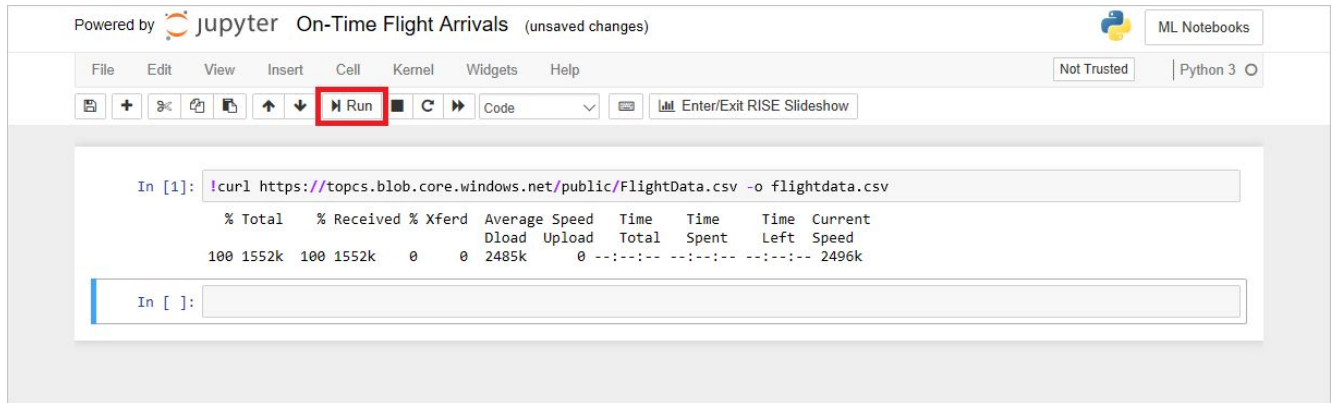
In this exercise, you will import a dataset from Azure blob storage and load it into the notebook. Jupyter notebooks are highly interactive, and since they can include executable code, they provide the perfect platform for manipulating data and building predictive models from it.

1. Enter the following command into the first cell of the notebook:

```
!curl https://topcs.blob.core.windows.net/public/FlightData.csv -o flightdata.csv
```

`curl` is a Bash command. You can execute Bash commands in a Jupyter notebook by prefixing them with an exclamation mark. This command downloads a CSV file from Azure blob storage and saves it using the name `flightdata.csv`.

2. Click the Run button to execute the `curl` command.





Importing a dataset

In the notebook's second cell, enter the following Python code to load `flightdata.csv`, create a [Pandas DataFrame](#) from it, and display the first five rows.

```
import pandas as pd
```

```
df = pd.read_csv('flightdata.csv')
```

3. `df.head()`
4. Click the Run button to execute the code. Confirm that the output resembles the output below.

Powered by  jupyter On-Time Flight Arrivals (unsaved changes)  ML Notebooks

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [1]: `!curl https://topcs.blob.core.windows.net/public/FlightData.csv -o flightdata.csv`

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
100 1552k  100 1552k    0     0  2485k    0 --:--:-- --:--:-- --:--:-- 2496k
```

In [2]: `import pandas as pd`

```
df = pd.read_csv('flightdata.csv')
df.head()
```

Out[2]:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NUM	FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	...	CRS_ARR_
0	2016	1	1	1	5	DL	N836DN	1399	10397	ATL	...	
1	2016	1	1	1	5	DL	N964DN	1476	11433	DTW	...	
2	2016	1	1	1	5	DL	N813DN	1597	10397	ATL	...	
3	2016	1	1	1	5	DL	N587NW	1768	14747	SEA	...	
4	2016	1	1	1	5	DL	N836DN	1823	14747	SEA	...	

5 rows x 26 columns

In []:

Loading the dataset

The DataFrame that you created contains on-time arrival information for a major U.S. airline. It has more than 11,000 rows and 26 columns. (The output says "5 rows" because DataFrame's `head` function only returns the first five rows.) Each row represents one flight and contains information such as the origin, the destination, the scheduled departure time, and whether the flight arrived on time or late. You will learn more about the data, including its content and structure, in the next lab.

5. Use the File -> Save and Checkpoint command to save the notebook.

Use the horizontal scroll bar to scroll left and right and view all the columns in the dataset. How many columns does the dataset contain? Can you guess what each column represents from the column names?

Overview- Part 2: Process

In the real world, few datasets can be used, as is to train machine-learning models. It is not uncommon for data scientists to spend 80% or more of their time on a project cleaning, preparing, and shaping the data — a process sometimes referred to as *data wrangling*. Typical actions include removing duplicate rows, removing rows or columns with missing values or algorithmically replacing the missing values, normalizing data, and selecting feature columns. A machine-learning model is only as good as the data it is trained with. Preparing the data is arguably the most crucial step in the machine-learning process.

In this lab, you will use the Jupyter notebook to wrangle the dataset that you imported. You will use the [Python Data Analysis Library](#) (Pandas) to do the bulk of the work in transforming the data. The goal is to get the dataset ready to use in a machine-learning model, and to get first-hand experience with Pandas.



Exercise 3: Explore the data

Before you can prepare a dataset, you need to understand its content and structure. In the previous lab, you imported a dataset containing on-time arrival information for a major U.S. airline. That data included 26 columns and thousands of rows, with each row representing one flight and containing information such as the flight's origin, destination, and scheduled departure time. You also loaded the data into a Jupyter notebook and used a simple Python script to create a Pandas DataFrame from it.

A [DataFrame](#) is a two-dimensional labeled data structure. The columns in a DataFrame can be of different types, just like columns in a spreadsheet or database table. It is the most

commonly used object in Pandas. In this exercise, you will examine the DataFrame — and the data inside it — more closely.

1. Return to [Azure Notebooks](#) and to the notebook that you created in the previous lab. If you closed the notebook after the previous lab, use the Cell -> Run All to rerun the all of the cells in the notebook after opening it.

```
In [1]: !curl https://topcs.blob.core.windows.net/public/FlightData.csv -o flightdata.csv

% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left  Speed
100 1552k  100 1552k    0     0  2485k      0 --:--:-- --:--:-- --:--:-- 2496k

In [2]: import pandas as pd

df = pd.read_csv('flightdata.csv')
df.head()
```

Out[2]:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NUM	FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	...	CRS_ARR
0	2016	1	1	1	5	DL	N836DN	1399	10397	ATL	...	
1	2016	1	1	1	5	DL	N964DN	1476	11433	DTW	...	
2	2016	1	1	1	5	DL	N813DN	1597	10397	ATL	...	
3	2016	1	1	1	5	DL	N587NW	1768	14747	SEA	...	
4	2016	1	1	1	5	DL	N836DN	1823	14747	SEA	...	

5 rows x 26 columns

```
In [ ]:
```

The FlightData notebook

2. The code that you added to the notebook in the previous lab creates a DataFrame from flightdata.csv and calls `DataFrame.head` on it to display the first five rows. One of the first things you typically want to know about a dataset is how many rows it contains. To get a count, type the following statement into an empty cell at the end of the notebook and run it:

```
df.shape
```

Confirm that the DataFrame contains 11,231 rows and 26 columns:

```
In [3]: df.shape
```

```
Out[3]: (11231, 26)
```

Getting a row and column count

3. Now take a moment to examine the 26 columns in the dataset. They contain important information such as the date that the flight took place (YEAR, MONTH, and DAY_OF_MONTH), the origin and destination (ORIGIN and DEST), the scheduled departure and arrival times (CRS_DEP_TIME and CRS_ARR_TIME), the difference between the scheduled arrival time and the actual arrival time in minutes (ARR_DELAY), and whether the flight was late by 15 minutes or more (ARR_DEL15).

Here is a complete list of the columns in the dataset. Times are expressed in 24-hour military time. For example, 1130 equals 11:30 a.m. and 1500 equals 3:00 p.m.

Column	Description
YEAR	Year that the flight took place
QUARTER	Quarter that the flight took place (1-4)
MONTH	Month that the flight took place (1-12)
DAY_OF_MONTH	Day of the month that the flight took place (1-31)

DAY_OF_WEEK	Day of the week that the flight took place (1=Monday, 2=Tuesday, etc.)
UNIQUE_CARRIER	Airline carrier code (e.g., DL)
TAIL_NUM	Aircraft tail number
FL_NUM	Flight number
ORIGIN_AIRPORT_ID	ID of the airport of origin
ORIGIN	Origin airport code (ATL, DFW, SEA, etc.)
DEST_AIRPORT_ID	ID of the destination airport
DEST	Destination airport code (ATL, DFW, SEA, etc.)
CRS_DEP_TIME	Scheduled departure time

DEP_TIME	Actual departure time
DEP_DELAY	Number of minutes departure was delayed
DEP_DEL15	0=Departure delayed less than 15 minutes 1=Departure delayed 15 minutes or more
CRS_ARR_TIME	Scheduled arrival time
ARR_TIME	Actual arrival time
ARR_DELAY	Number of minutes flight arrived late
ARR_DEL15	0=Arrived less than 15 minutes late 1=Arrived 15 minutes or more late
CANCELLED	0=Flight was not cancelled 1=Flight was cancelled

DIVERTED	0=Flight was not diverted 1=Flight was diverted
CRS_ELAPSED_TIME	Scheduled flight time in minutes
ACTUAL_ELAPSED_TIME	Actual flight time in minutes
DISTANCE	Distance traveled in miles

The dataset includes a roughly even distribution of dates throughout the year, which is important because a flight out of Minneapolis is less likely to be delayed due to winter storms in July than it is in January. But this dataset is far from being "clean" and ready to use. Let's write some Pandas code to clean it up.

Exercise 4: Clean the data

One of the most important aspects of preparing a dataset for use in machine learning is selecting the "feature" columns that are relevant to the outcome you are trying to predict while filtering out columns that do not affect the outcome, could bias it in a negative way, or might produce [multicollinearity](#). Another important task is to eliminate missing values, either by deleting the rows or columns containing them or replacing them with meaningful values. In this exercise, you will eliminate extraneous columns and replace missing values in the remaining columns.

1. One of the first things data scientists typically look for in a dataset is missing values. There's an easy way to check for missing values in Pandas. To demonstrate, execute

the following code in a cell at the end of the notebook:

```
df.isnull().values.any()
```

Confirm that the output is "True," which indicates that there is at least one missing value somewhere in the dataset.

```
In [4]: df.isnull().values.any()
```

```
Out[4]: True
```

Checking for missing values

2. The next step is to find out where the missing values are. To do so, execute the following code:

```
df.isnull().sum()
```

Confirm that you see the following output listing a count of missing values in each column:

YEAR	0
QUARTER	0
MONTH	0
DAY_OF_MONTH	0
DAY_OF_WEEK	0
UNIQUE_CARRIER	0
TAIL_NUM	0
FL_NUM	0
ORIGIN_AIRPORT_ID	0
ORIGIN	0
DEST_AIRPORT_ID	0
DEST	0
CRS_DEP_TIME	0
DEP_TIME	107
DEP_DELAY	107
DEP_DEL15	107
CRS_ARR_TIME	0
ARR_TIME	115
ARR_DELAY	188
ARR_DEL15	188
CANCELLED	0
DIVERTED	0
CRS_ELAPSED_TIME	0
ACTUAL_ELAPSED_TIME	188
DISTANCE	0
Unnamed: 25	11231

dtype: int64

Number of missing values in each column

Curiously, the 26th column ("Unnamed: 25") contains 11,231 missing values, which equals the number of rows in the dataset. This column was mistakenly created because the CSV file that you imported contains a comma at the end of each line. To eliminate that column, add the following code to the notebook and execute it:

```
df = df.drop('Unnamed: 25', axis=1)
```

3. `df.isnull().sum()`

Inspect the output and confirm that column 26 has disappeared from the DataFrame:

YEAR	0
QUARTER	0
MONTH	0
DAY_OF_MONTH	0
DAY_OF_WEEK	0
UNIQUE_CARRIER	0
TAIL_NUM	0
FL_NUM	0
ORIGIN_AIRPORT_ID	0
ORIGIN	0
DEST_AIRPORT_ID	0
DEST	0
CRS_DEP_TIME	0
DEP_TIME	107
DEP_DELAY	107
DEP_DEL15	107
CRS_ARR_TIME	0
ARR_TIME	115
ARR_DELAY	188
ARR_DEL15	188
CANCELLED	0
DIVERTED	0
CRS_ELAPSED_TIME	0
ACTUAL_ELAPSED_TIME	188
DISTANCE	0

dtype: int64

The DataFrame with column 26 removed

The DataFrame still contains a lot of missing values, but some of them are irrelevant because the columns containing them are not germane to the model that you are building. The goal of that model is to predict whether a flight you are considering booking is likely to arrive on time. If you know that the flight is likely to be late, you might choose to book another flight. The next step, therefore, is to filter the dataset to eliminate columns that aren't relevant to a predictive model. For example, the aircraft's tail number probably has little bearing on whether a flight will arrive on time, and at the time you book a ticket, you have no way of knowing whether a flight will be cancelled, diverted, or delayed. By contrast, the scheduled departure time could have a *lot* to do with on-time arrivals. Because of the hub-and-spoke system used by most airlines, morning flights tend to be on time more often than afternoon or evening flights. And at some major airports, traffic stacks up during the day, increasing the likelihood that later flights will be delayed.

Pandas provides an easy way to filter out columns you don't want. Execute the following code in a new cell at the end of the notebook:

```
df = df[["MONTH", "DAY_OF_MONTH", "DAY_OF_WEEK", "ORIGIN", "DEST", "CRS_DEP_TIME", "ARR_DEL15"]]
```

4. `df.isnull().sum()`

The output shows that the DataFrame now includes only the columns that are relevant to the model, and that the number of missing values is greatly reduced:

```
MONTH          0
DAY_OF_MONTH   0
DAY_OF_WEEK    0
ORIGIN         0
DEST           0
CRS_DEP_TIME   0
ARR_DEL15     188
dtype: int64
```

The filtered DataFrame

5. The only column that now contains missing values is the ARR_DEL15 column, which uses 0s to identify flights that arrived on time and 1s for flights that didn't. Use the following code to show the first five rows with missing values:

```
df[df.isnull().values.any(axis=1)].head()
```

Pandas represents missing values with `NaN`, which stands for *Not a Number*. The output shows that these rows are indeed missing values in the ARR_DEL15 column:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
177	1	9	6	MSP	SEA	701	NaN
179	1	10	7	MSP	DTW	1348	NaN
184	1	10	7	MSP	DTW	625	NaN
210	1	10	7	DTW	MSP	1200	NaN
478	1	22	5	SEA	JFK	2305	NaN

Rows with missing values

The reason these rows are missing ARR_DEL15 values is that they all correspond to flights that were canceled or diverted. You could call `dropna` on the DataFrame to remove these rows. But since a flight that is canceled or diverted to another airport could be considered

"late," let's use the `fillna` method to replace the missing values with 1s.

Use the following code to replace missing values in the `ARR_DEL15` column with 1s and display rows 177 through 184:

```
df = df.fillna({'ARR_DEL15': 1})
```

6. `df.iloc[177:185]`

Confirm that the `NaN`s in rows 177, 179, and 184 were replaced with 1s indicating that the flights arrived late:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
177	1	9	6	MSP	SEA	701	1.0
178	1	9	6	DTW	JFK	1527	0.0
179	1	10	7	MSP	DTW	1348	1.0
180	1	10	7	DTW	MSP	1540	0.0
181	1	10	7	JFK	ATL	1325	0.0
182	1	10	7	JFK	ATL	610	0.0
183	1	10	7	JFK	SEA	1615	0.0
184	1	10	7	MSP	DTW	625	1.0

NaNs replaced with 1s

The dataset is now "clean" in the sense that missing values have been replaced and the list of columns has been narrowed to those most relevant to the model. But you're not finished yet. There is more to do to prepare the dataset for use in machine learning.

Exercise 5: Bin departure times and add indicator columns

The `CRS_DEP_TIME` column of the dataset you are using represents scheduled departure times. The granularity of the numbers in this column — it contains more than 500 unique values — could have a negative impact on accuracy in a machine-learning model. This can be resolved using a technique called `binning` or quantization. What if you divided each number in this column by 100 and rounded down to the nearest integer? 1030 would become 10, 1925 would become 19, and so on, and you would be left with a maximum of 24 discrete

values in this column. Intuitively, it makes sense, because it probably doesn't matter much whether a flight leaves at 10:30 a.m. or 10:40 a.m. It matters a great deal whether it leaves at 10:30 a.m. or 5:30 p.m.

In addition, the dataset's `ORIGIN` and `DEST` columns contain airport codes that represent categorical machine-learning values. These columns need to be converted into discrete columns containing indicator variables, sometimes known as "dummy" variables. In other words, the `ORIGIN` column, which contains five airport codes, needs to be converted into five columns, one per airport, with each column containing 1s and 0s indicating whether a flight originated at the airport that the column represents. The `DEST` column needs to be handled in a similar manner.

In this exercise, you will "bin" the departure times in the `CRS_DEP_TIME` column and use Pandas' `get_dummies` method to create indicator columns from the `ORIGIN` and `DEST` columns.

1. Use the following command to display the first five rows of the DataFrame:

```
df.head()
```

Observe that the `CRS_DEP_TIME` column contains values from 0 to 2359 representing military times.

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
0	1	1	5	ATL	SEA	1905	0.0
1	1	1	5	DTW	MSP	1345	0.0
2	1	1	5	ATL	SEA	940	0.0
3	1	1	5	SEA	MSP	819	0.0
4	1	1	5	SEA	DTW	2300	0.0

The DataFrame with unbinned departure times

Use the following statements to bin the departure times:

```
import math
```

```
for index, row in df.iterrows():
```

```
    df.loc[index, 'CRS_DEP_TIME'] = math.floor(row['CRS_DEP_TIME'] / 100)
```

2. `df.head()`

Confirm that the numbers in the CRS_DEP_TIME column now fall in the range 0 to 23:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
0	1	1	5	ATL	SEA	19	0.0
1	1	1	5	DTW	MSP	13	0.0
2	1	1	5	ATL	SEA	9	0.0
3	1	1	5	SEA	MSP	8	0.0
4	1	1	5	SEA	DTW	23	0.0

The DataFrame with binned departure times

Now use the following statements to generate indicator columns from the ORIGIN and DEST columns, while dropping the ORIGIN and DEST columns themselves:

```
df = pd.get_dummies(df, columns=['ORIGIN', 'DEST'])
```

3. `df.head()`

Examine the resulting DataFrame and observe that the ORIGIN and DEST columns were replaced with columns corresponding to the airport codes present in the original columns. The new columns have 1s and 0s indicating whether a given flight originated at or was destined for the corresponding airport.

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_DEP_TIME	ARR_DEL15	ORIGIN_ATL	ORIGIN_DTW	ORIGIN_JFK	ORIGIN
0	1	1	5	19	0.0	1	0	0	
1	1	1	5	13	0.0	0	1	0	
2	1	1	5	9	0.0	1	0	0	
3	1	1	5	8	0.0	0	0	0	
4	1	1	5	23	0.0	0	0	0	

The DataFrame with indicator columns

4. Use the File -> Save and Checkpoint command to save the notebook.

The dataset looks very different than it did at the start, but it is now optimized for use in machine learning.

Overview- Part 3: Predict

Machine learning's usefulness will only increase as more and more data becomes available and the desire to perform predictive analytics from that data grows, too.

One of the most popular tools for building machine-learning models is [Scikit-learn](#), a free and open-source toolkit for Python programmers. It has built-in support for popular regression, classification, and clustering algorithms and works with other Python libraries such as [NumPy](#) and [SciPy](#). Scikit-learn allows you to focus on building, training, tuning, and testing machine-learning models without getting bogged down with coding algorithms.

We will be using Scikit-learn to build a machine-learning model utilizing on-time arrival data for a major U.S. airline. The goal is to create a model that might be useful in the real world for predicting whether a flight is likely to arrive on time.



Exercise 6: Split the data

To create a machine-learning model, you need two datasets: one for training and one for testing. In practice, you often have only one dataset, so you split it into two datasets. In this exercise, you will perform an 80-20 split on the DataFrame you prepared in the previous lab so you can use it to train a machine-learning model. You will also separate the DataFrame into feature columns and label columns. The former contains the columns used as input to the model (for example, the flight's origin and destination and the scheduled departure time), while the latter contains the column that the model will attempt to predict — in this case, the `ARR_DEL15` column which indicates whether a flight will arrive on time.

1. Return to [Azure Notebooks](#) and to the notebook that you created in the first lab. If you closed the notebook after the previous lab, use the Cell -> Run All to rerun the all of the cells in the notebook after opening it.

In a new cell at the end of the notebook, enter and execute the following statements:

```
from sklearn.model_selection import train_test_split
```

2.

```
train_x, test_x, train_y, test_y = train_test_split(df.drop('ARR_DEL15', axis=1), df['ARR_DEL15'], test_size=0.2, random_state=42)
```

The first statement imports Sckit-learn's `train_test_split` helper function. The second line uses the function to split the DataFrame into a training set containing 80% of the original data, and a test set containing the remaining 20%. The `random_state` parameter seeds the random-number generator used to do the splitting, while the first and second parameters are DataFrames containing the feature columns and the label column.

3. `train_test_split` returns four DataFrames. Use the following command to display the number of rows and columns in the DataFrame containing the feature columns used for training:

```
train_x.shape
```

4. Now use this command to display the number of rows and columns in the DataFrame containing the feature columns used for testing:

```
test_x.shape
```

How do the two outputs differ, and why?

Can you predict what you would see if you called `shape` on the other two DataFrames, `train_y` and `test_y`? If you're not sure, try it and find out.

Exercise 7: Train a machine-learning model

There are many types of machine-learning models. One of the most common is the regression model, which uses one of a number of regression algorithms to produce a numeric value — for example, a person's age or the probability that a credit-card transaction is fraudulent. You will train a classification model, which seeks to resolve a set of inputs into one of a set of known outputs. A classic example of a classification model is one that examines e-mails and classifies them as "spam" or "not spam." Your model will be a binary classification model that predicts whether a flight will arrive on-time or late ("binary" because there are only two possible outputs).

One of the benefits of using Sckit-learn is that you don't have to build these models — or implement the algorithms that they use — by hand. Sckit-learn includes a variety of classes for implementing common machine-learning models. One of them is [RandomForestClassifier](#), which fits multiple decision trees to the data and uses averaging to boost the overall accuracy and limit [overfitting](#).

Execute the following code in a new cell to create a `RandomForestClassifier` object and train it by calling the `fit` method.

```
from sklearn.ensemble import RandomForestClassifier
```

```
model = RandomForestClassifier(random_state=13)
```

1. `model.fit(train_x, train_y)`

The output shows the parameters used in the classifier, including `n_estimators`, which specifies the number of trees in each decision-tree forest, and `max_depth`, which specifies the maximum depth of the decision trees. The values shown are the defaults, but you can override any of them when creating the `RandomForestClassifier` object.

```
In [16]: from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=13)
model.fit(train_x, train_y)

Out[16]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=None, max_features='auto', max_leaf_nodes=None,
                                min_impurity_split=1e-07, 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=13,
                                verbose=0, warm_start=False)
```

Training the model

Now call the `predict` method to test the model using the values in `test_x`, followed by the `score` method to determine the mean accuracy of the model:

```
predicted = model.predict(test_x)
```

2. `model.score(test_x, test_y)`

Confirm that you see the following output:

```
In [17]: predicted = model.predict(test_x)
         model.score(test_x, test_y)

Out[17]: 0.86025812194036488
```

Testing the model

The mean accuracy is 86%, which seems good on the surface. However, mean accuracy isn't always a reliable indicator of the accuracy of a classification model. Let's dig a little deeper and determine how accurate the model really is — that is, how adept it is at determining whether a flight will arrive on time.

Exercise 8: Gauge the model's accuracy

There are several ways to measure the accuracy of a classification model. One of the best overall measures for a binary classification model is [Area Under Receiver Operating Characteristic Curve](#) (sometimes referred to as "ROC AUC"), which essentially quantifies how often the model will make a correct prediction regardless of the outcome. In this exercise, you will compute an ROC AUC score for the model you built in the previous exercise and learn about some of the reasons why that score is lower than the mean accuracy output by the `score` method. You will also learn about other ways to gauge the accuracy of the model.

Before you compute the ROC AUC, you must generate *prediction probabilities* for the test set. These probabilities are estimates for each of the classes, or answers, the model can predict. For example, `[0.88199435, 0.11800565]` means that there's an 89% chance that a flight will arrive on time (`ARR_DEL15 = 0`) and a 12% chance that it won't (`ARR_DEL15 = 1`). The sum of the two probabilities adds up to 100%.

Run the following code to generate a set of prediction probabilities from the test data:

```
from sklearn.metrics import roc_auc_score
```

1. `probabilities = model.predict_proba(test_x)`
2. Now use the following statement to generate an ROC AUC score from the probabilities using Sckit-learn's `roc_auc_score` method:

```
roc_auc_score(test_y, probabilities[:, 1])
```

Confirm that the output shows a score of 67%:

```
In [19]: roc_auc_score(test_y, probabilities[:, 1])
```

```
Out[19]: 0.67438249049985388
```

Generating an AUC score

Why is the AUC score lower than the mean accuracy computed in the previous exercise?

The output from the `score` method reflects how many of the items in the test set the model predicted correctly. This score is skewed by the fact that the dataset the model was trained and tested with contains many more rows representing on-time arrivals than rows representing late arrivals. Because of this imbalance in the data, you are more likely to be correct if you predict that a flight will be on time than if you predict that a flight will be late.

ROC AUC takes this into account and provides a more accurate indication of how likely it is that a prediction of on-time *or* late will be correct.

You can learn more about the model's behavior by generating a [confusion matrix](#), also known as an *error matrix*. The confusion matrix quantifies the number of times each answer was classified correctly or incorrectly. Specifically, it quantifies the number of false positives, false negatives, true positives, and true negatives. This is important, because if a binary classification model trained to recognize cats and dogs is tested with a dataset that is 95% dogs, it could score 95% simply by guessing "dog" every time. But if it failed to identify cats at all, it would be of little value.

Use the following code to produce a confusion matrix for your model:

```
from sklearn.metrics import confusion_matrix
```

```
3. confusion_matrix(test_y, predicted)
```

The first row in the output represents flights that were on time. The first column in that row shows how many flights were correctly predicted to be on time, while the second column reveals how many flights were predicted as delayed but were not. From this, the model appears to be very adept at predicting that a flight will be on time.

```
In [20]: from sklearn.metrics import confusion_matrix
         confusion_matrix(test_y, predicted)
```

```
Out[20]: array([[1882,  54],
               [ 260,  51]])
```

Generating a confusion matrix

But look at the second row, which represents flights that were delayed. The first column shows how many delayed flights were incorrectly predicted to be on time. The second column shows how many flights were correctly predicted to be delayed. Clearly, the model isn't nearly as adept at predicting that a flight will be delayed as it is at predicting that a flight will arrive on time. What you *want* in a confusion matrix is big numbers in the upper-left and lower-right corners, and small numbers (preferably zeros) in the upper-right and lower-left corners.

Other measures of accuracy for a classification model include *precision* and *recall*. Suppose the model was presented with three on-time arrivals and three delayed arrivals, and that it correctly predicted two of the on-time arrivals, but incorrectly predicted that two of the delayed arrivals would be on time. In this case, the precision would be 50% (two of the four flights it classified as being on time actually were on time), while its recall would be 67% (it correctly identified two of the three on-time arrivals). You can learn more about precision and recall from https://en.wikipedia.org/wiki/Precision_and_recall

Scikit-learn contains a handy method named `precision_score` for computing precision. To quantify the precision of your model, execute the following statements:

```
from sklearn.metrics import precision_score
```

```
train_predictions = model.predict(train_x)
```

4. `precision_score(train_y, train_predictions)`

Examine the output. What is your model's precision?

```
In [21]: from sklearn.metrics import precision_score

train_predictions = model.predict(train_x)
precision_score(train_y, train_predictions)
```

```
Out[21]: 0.88750000000000006
```

Measuring precision

Scikit-learn also contains a method named `recall_score` for computing recall. To measure your model's recall, execute the following statements:

```
from sklearn.metrics import recall_score
```

5. `recall_score(train_y, train_predictions)`

What is the model's recall?

```
In [22]: from sklearn.metrics import recall_score

recall_score(train_y, train_predictions)
```

```
Out[22]: 0.88461538461538464
```

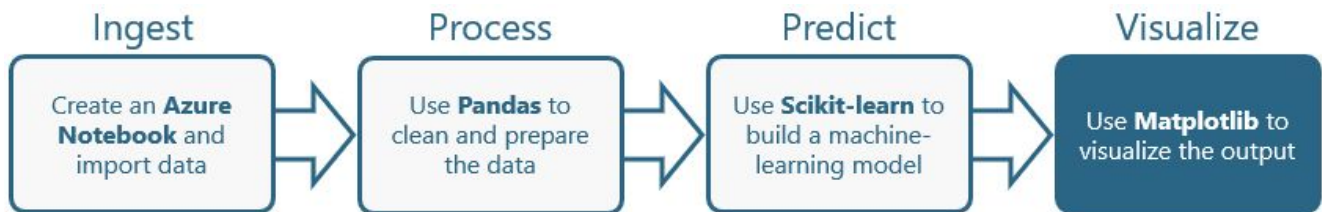
Measuring recall

6. Use the File -> Save and Checkpoint command to save the notebook.

In the real world, a trained data scientist would look for ways to make the model even more accurate. Among other things, he or she would try different algorithms and take steps to *tune* the chosen algorithm to find the optimum combination of parameters. Another likely step would be to expand the dataset to millions of rows rather than a few thousand and also attempt to reduce the imbalance between late and on-time arrivals. But for our purposes, the model is fine as is.

Overview- Part 4: Visualize

Now that you have trained a machine-learning model to perform predictive analytics, it's time to put it to work. You are going to write a function that uses the machine-learning model that you just built to predict whether a flight will arrive on time or late. And you will use [Matplotlib](#), a popular plotting and charting library for Python, to visualize the results.



Exercise 9: Import Matplotlib

In this exercise, you will import Matplotlib into the notebook you have been working with and configure the notebook to support inline Matplotlib output.

1. Return to [Azure Notebooks](#) and to the notebook that you created in the first lab. If you closed the notebook after the previous lab, use the Cell -> Run All to rerun all of the cells in the notebook after opening it.

Execute the following statements in a new cell at the end of the notebook. Ignore any warning messages that are displayed related to font caching:

```
%matplotlib inline
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

2. `sns.set()`

The first statement is one of several [magic commands](#) supported by the Python kernel that you selected when you created the notebook. It enables Jupyter to render

Matplotlib output in a notebook without making repeated calls to [show](#). And it must appear before any references to Matplotlib itself. The final statement configures [Seaborn](#) to enhance the output from Matplotlib.

To see Matplotlib at work, execute the following code in a new cell to plot the [ROC curve](#) for the machine-learning model you built in the previous lab:

```
from sklearn.metrics import roc_curve

fpr, tpr, _ = roc_curve(test_y, probabilities[:, 1])

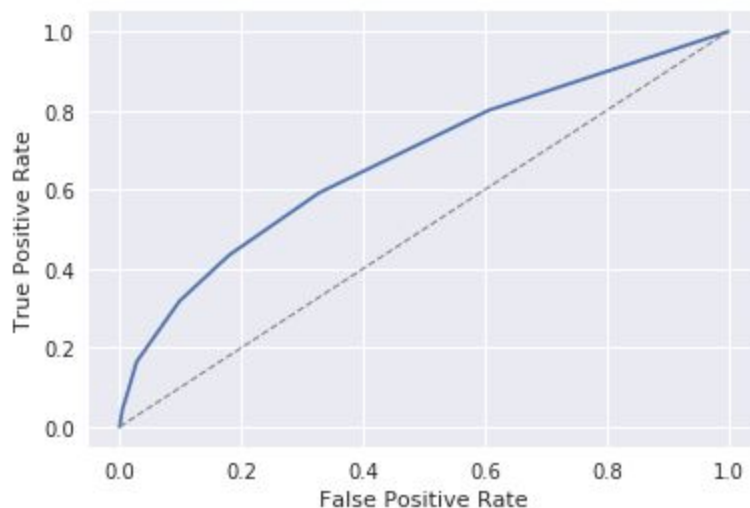
plt.plot(fpr, tpr)

plt.plot([0, 1], [0, 1], color='grey', lw=1, linestyle='--')

plt.xlabel('False Positive Rate')

3. plt.ylabel('True Positive Rate')
```

4. Confirm that you see the following output:



ROC curve generated with Matplotlib

The dotted line in the middle of the graph represents a 50-50 chance of obtaining a correct answer. The blue curve represents the accuracy of your model. More importantly, the fact that this chart appears at all demonstrates that you can use Matplotlib in a Jupyter notebook.

Exercise 10: Predict on-time arrivals

In this exercise, you will write a Python function that calls the machine-learning model you built in the previous lab to compute the likelihood that a flight will be on time. Then you will use the function to analyze several flights.

Enter the following function definition in a new cell, and then run the cell.

```
def predict_delay(departure_date_time, origin, destination):  
  
    from datetime import datetime  
  
  
    try:  
  
        departure_date_time_parsed = datetime.strptime(departure_date_time,  
        '%d/%m/%Y %H:%M:%S')  
  
    except ValueError as e:  
  
        return 'Error parsing date/time - {}'.format(e)  
  
  
    month = departure_date_time_parsed.month  
  
    day = departure_date_time_parsed.day  
  
    day_of_week = departure_date_time_parsed.isoweekday()  
  
    hour = departure_date_time_parsed.hour  
  
  
    origin = origin.upper()
```

```
destination = destination.upper()
```

```
input = [{'MONTH': month,
          'DAY': day,
          'DAY_OF_WEEK': day_of_week,
          'CRS_DEP_TIME': hour,
          'ORIGIN_ATL': 1 if origin == 'ATL' else 0,
          'ORIGIN_DTW': 1 if origin == 'DTW' else 0,
          'ORIGIN_JFK': 1 if origin == 'JFK' else 0,
          'ORIGIN_MSP': 1 if origin == 'MSP' else 0,
          'ORIGIN_SEA': 1 if origin == 'SEA' else 0,
          'DEST_ATL': 1 if destination == 'ATL' else 0,
          'DEST_DTW': 1 if destination == 'DTW' else 0,
          'DEST_JFK': 1 if destination == 'JFK' else 0,
          'DEST_MSP': 1 if destination == 'MSP' else 0,
          'DEST_SEA': 1 if destination == 'SEA' else 0 }]
```

```
1. return model.predict_proba(pd.DataFrame(input))[0][0]
```

This function takes as input a date and time, an origin airport code, and a destination airport code, and returns a value between 0.0 and 1.0 indicating the probability that the flight will arrive at its destination on time. It uses the machine-learning model you built in the previous lab to compute the probability. And to call the model, it passes a DataFrame containing the input values to `predict_proba`. The structure of the DataFrame exactly matches the structure of the DataFrame depicted in Exercise 3, Step 3 of [Lab 2](#).

Note that dates input to the `predict_delay` function use the international date format dd/mm/year.

2. Use the code below to compute the probability that a flight from New York to Atlanta on the evening of October 1 will arrive on time. Note that the year you enter is irrelevant because it isn't used by the model.

```
predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL')
```

Confirm that the output shows that the likelihood of an on-time arrival is 60%:

```
In [26]: predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL')
```

```
Out[26]: 0.5999999999999998
```

Predicting whether a flight will arrive on time

3. Modify the code to compute the probability that the same flight a day later will arrive on time:

```
predict_delay('2/10/2018 21:45:00', 'JFK', 'ATL')
```

How likely is this flight to arrive on time? If your travel plans were flexible, would you consider postponing your trip for one day?

4. Now modify the code to compute the probability that a morning flight the same day from Atlanta to Seattle will arrive on time:

```
predict_delay('2/10/2018 10:00:00', 'ATL', 'SEA')
```

Is this flight likely to arrive on time?

You now have an easy way to predict, with a single line of code, whether a flight is likely to be on time or late. Feel free to experiment with other dates, times, origins, and destinations. But keep in mind that the results are only meaningful for the airport codes ATL, DTW, JFK, MSP, and SEA because those are the only airport codes the model was trained with.

Exercise 11: Plot predictions

In this exercise, you will combine the `predict_delay` function you created in the previous exercise with Matplotlib to produce side-by-side comparisons of the same flight on consecutive days and flights with the same origin and destination at different times throughout the day.

Execute the following code to plot the probability of on-time arrivals for an evening flight from JFK to ATL over a range of days:

```
import numpy as np

labels = ('Oct 1', 'Oct 2', 'Oct 3', 'Oct 4', 'Oct 5', 'Oct 6', 'Oct 7')

values = (predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('2/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('3/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('4/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('5/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('6/10/2018 21:45:00', 'JFK', 'ATL'),
          predict_delay('7/10/2018 21:45:00', 'JFK', 'ATL'))

alabels = np.arange(len(labels))

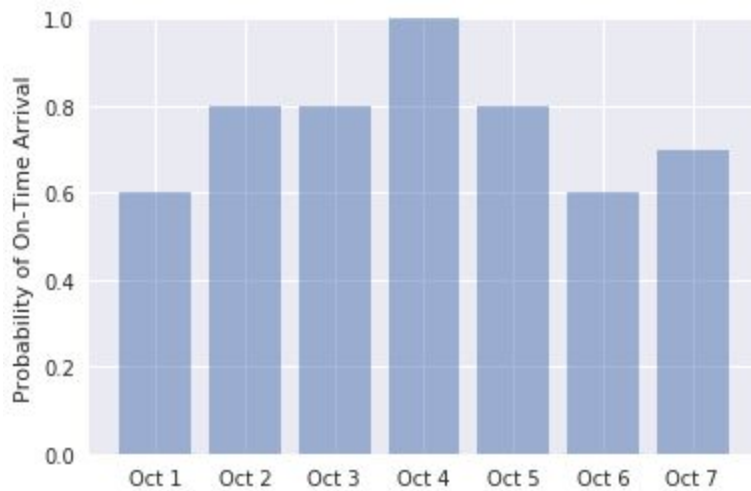
plt.bar(alabels, values, align='center', alpha=0.5)

plt.xticks(alabels, labels)

plt.ylabel('Probability of On-Time Arrival')

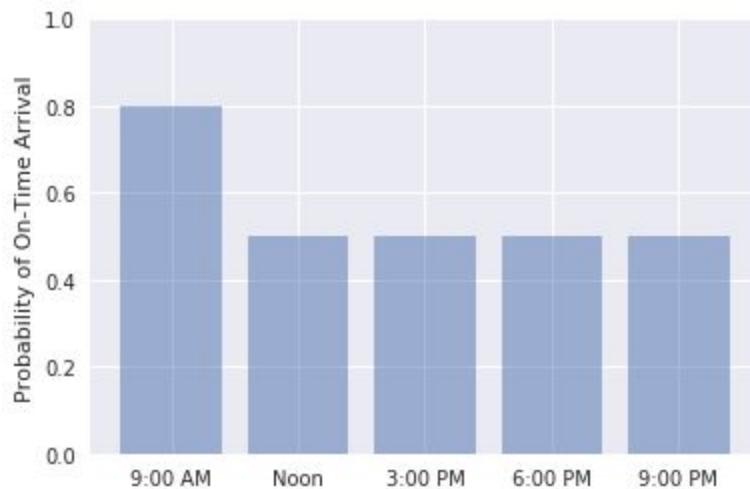
1. plt.ylim((0.0, 1.0))
```

2. Confirm that the output looks like this:



Probability of on-time arrivals for a range of dates

3. Modify the code to produce a similar chart for flights leaving JFK for MSP at 1:00 p.m. on April 10 through April 16. How does the output compare to the output in the previous step?
4. On your own, write code to graph the probability that flights leaving SEA for ATL at 9:00 a.m., noon, 3:00 p.m., 6:00 p.m., and 9:00 p.m. on January 30 will arrive on time. Confirm that the output matches this:



Probability of on-time arrivals for a range of times

If you are new to Matplotlib and would like to learn more about it, you will find an excellent tutorial at <https://www.labri.fr/perso/nrougier/teaching/matplotlib/>. There is *much* more to Matplotlib than what was shown here, which is one reason why it is so popular in the Python community.

Summary

In four hands-on labs, you learned how to:

- Create a notebook in Azure Notebooks
- Import data into a notebook using `curl`
- Use [Pandas](#) to clean and prepare data
- Use [Scikit-learn](#) to build a machine-learning model
- Use [Matplotlib](#) to visualize the results

Pandas, Scikit-learn, and Matplotlib are among the most popular Python libraries on the planet. With them, you can prepare data for use in machine learning, build sophisticated machine-learning models from the data, and chart the output. Jupyter notebooks provide a ready-made environment for using these libraries, and Azure Notebooks give you easy

access to Jupyter notebooks without requiring you to install any software or set up a Jupyter environment on a server.

Sources:

<https://github.com/microsoft/computerscience/tree/master/Labs/Deep%20Learning/200%20-%20Machine%20Learning%20in%20Python>