ENTERPRISE ARTIFICIAL INTELLIGENCE TRANSFORMATION

BY RASHED HAQ

Contents

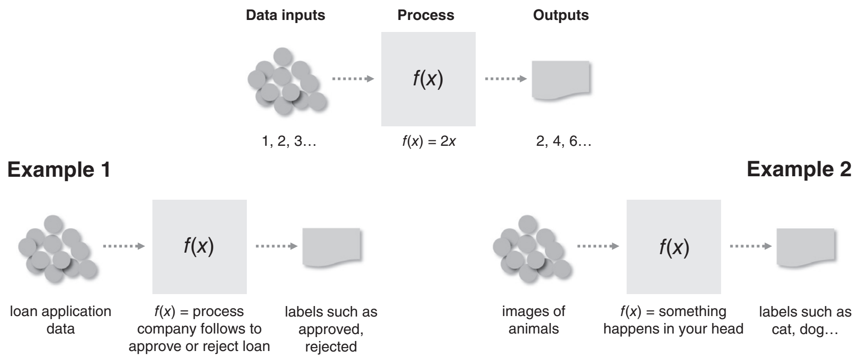

Figure 2.1 Examples of functions $f(x)$ that can be estimated by using machine learning on the input and output datasets.

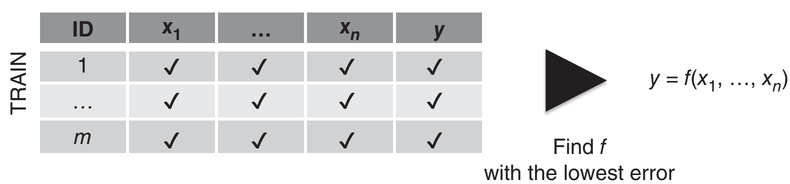

Figure 2.2 Using training data for customers 1 to m to estimate f that will predict y given $x_1, \ldots, x_n$.

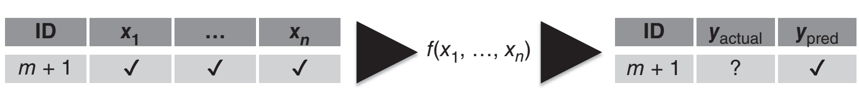

Figure 2.3 Using the machine-learning model (f) to predict if customer number $m + 1$ will churn.

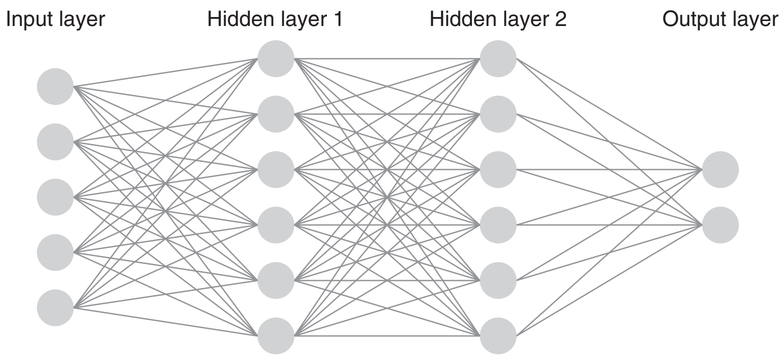

Figure 2.4 An example of a deep neural network.

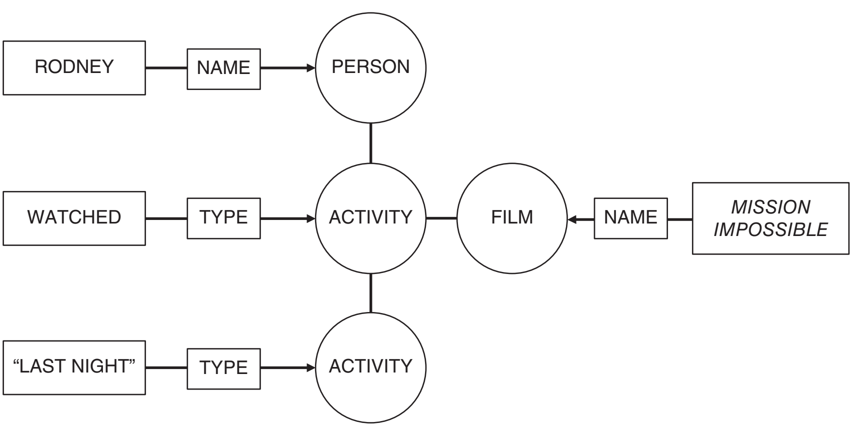

Figure 2.5 Example of a type of knowledge graph.

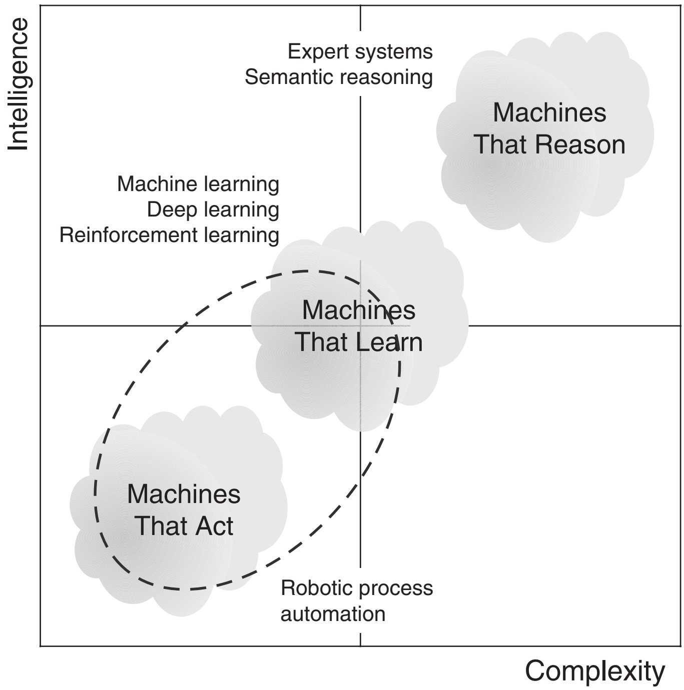

Figure 2.6 Types of AI systems.

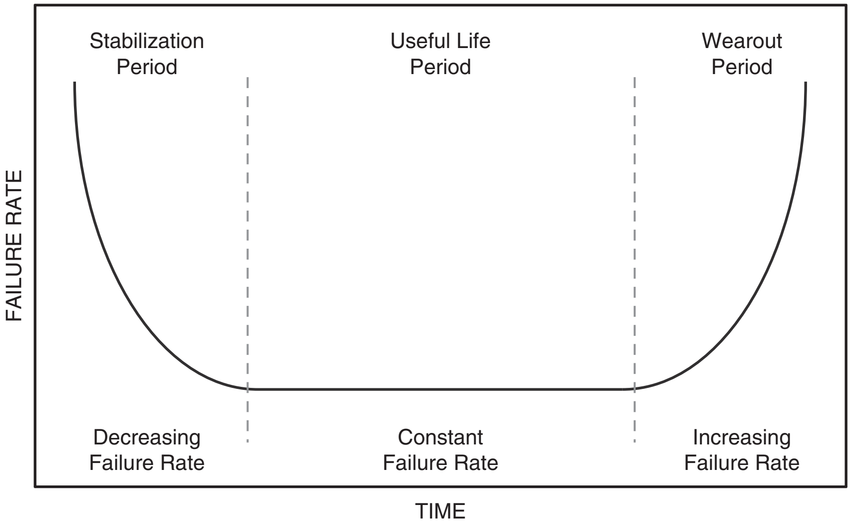

Figure 5.1 Heuristic showing different failure rates during equipment component lifecycle.

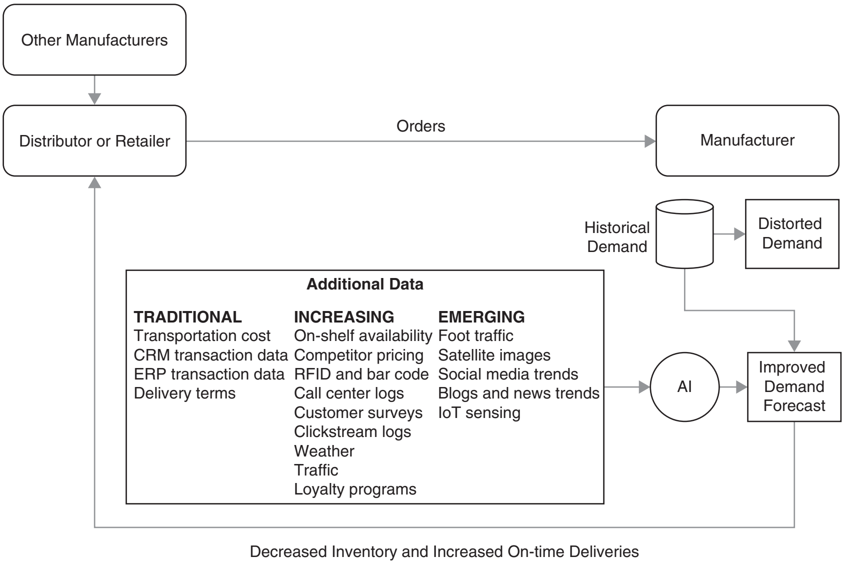

Figure 5.2 Demand forecasting using historical sales and new data sources.

Figure 5.3 Energy trading scenario.

Firmographic	Demographic	Technographic	Relationship	Satellite	News and Blogs	Device	Behavioral
Revenues	Biography	IT infrastructure	Social network	Foot traffic	News sites	Reverse IP address	Content consumption
Employees	Title/role	Applications (used or installed)	Reporting relationship	Parking lot usage	Analyst and broker reports	Geolocation	Search history
SIC/NAICS codes	Time in title/role	Professional services contracts	External social contacts	Inventory and movement	Blogs	Usage	Site, page, and app visits
Office locations	Office Location	Equipment	Internal colleague and employee contacts				Social posts
Department and business unit	Department	Other technologies	Influence rating (Klout or proprietary score)				
Press releases	Education	Contract renewal date					
Job postings							

Figure 7.1 Different types of third-party data that is available commercially or through the web.

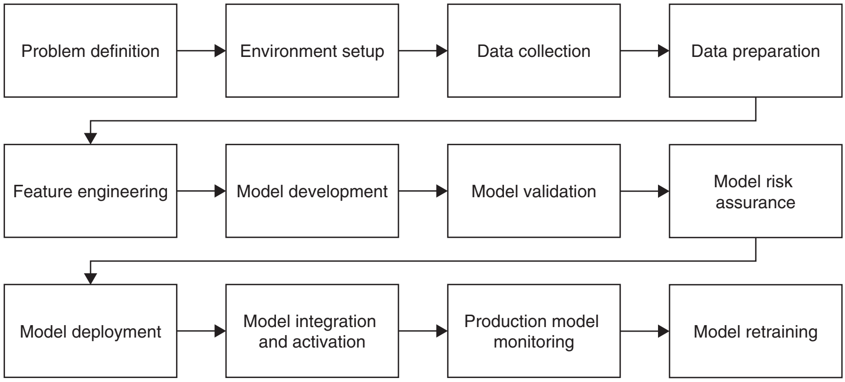

Figure 8.1 The workflow for AI, machine learning, and data science projects.

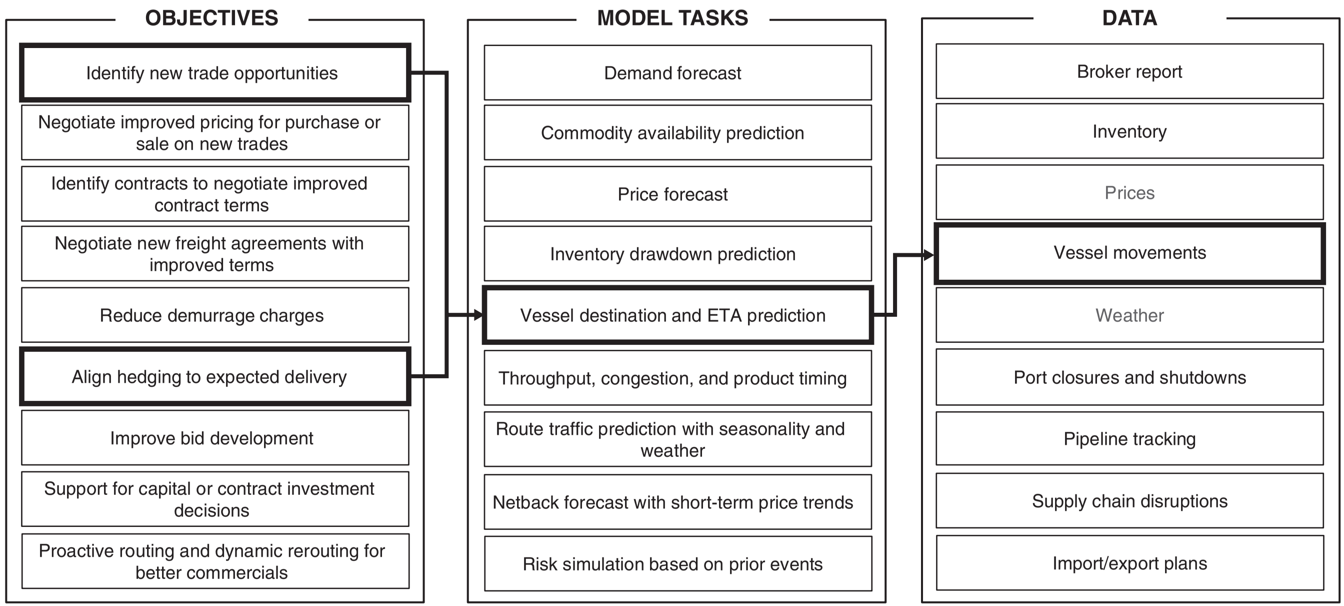

Figure 8.2 A sample map of use case objectives, modeling tasks to support the decisions, and data required to support modeling.

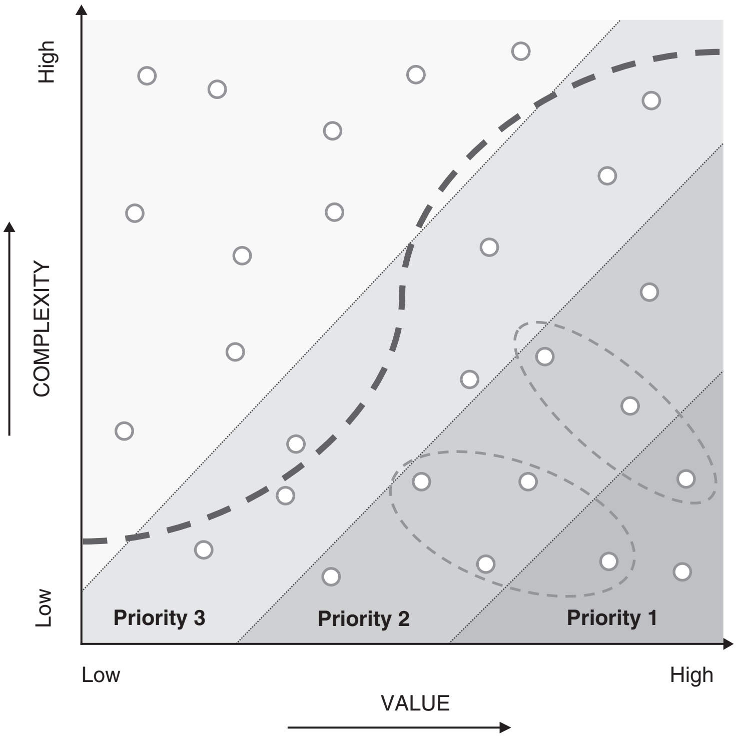

Figure 8.3 Graph showing use cases by value and complexity.

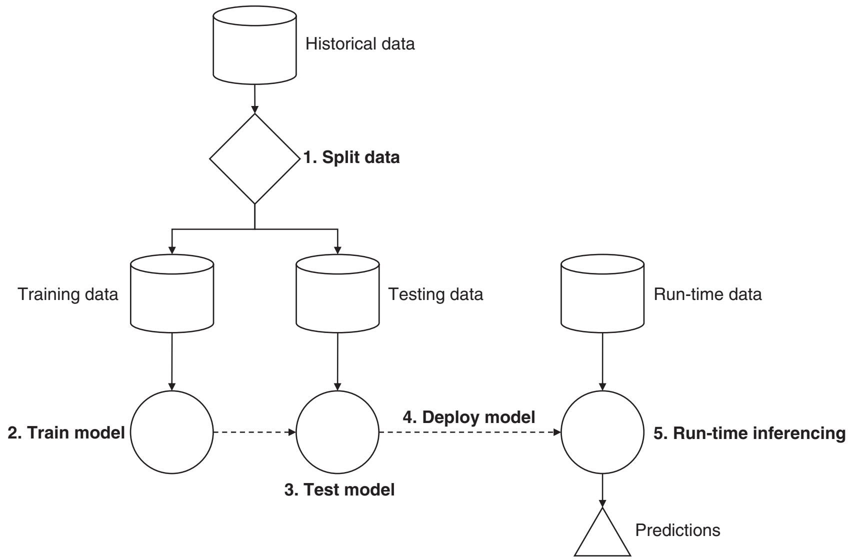

Figure 8.4 Process for training and validating the model.

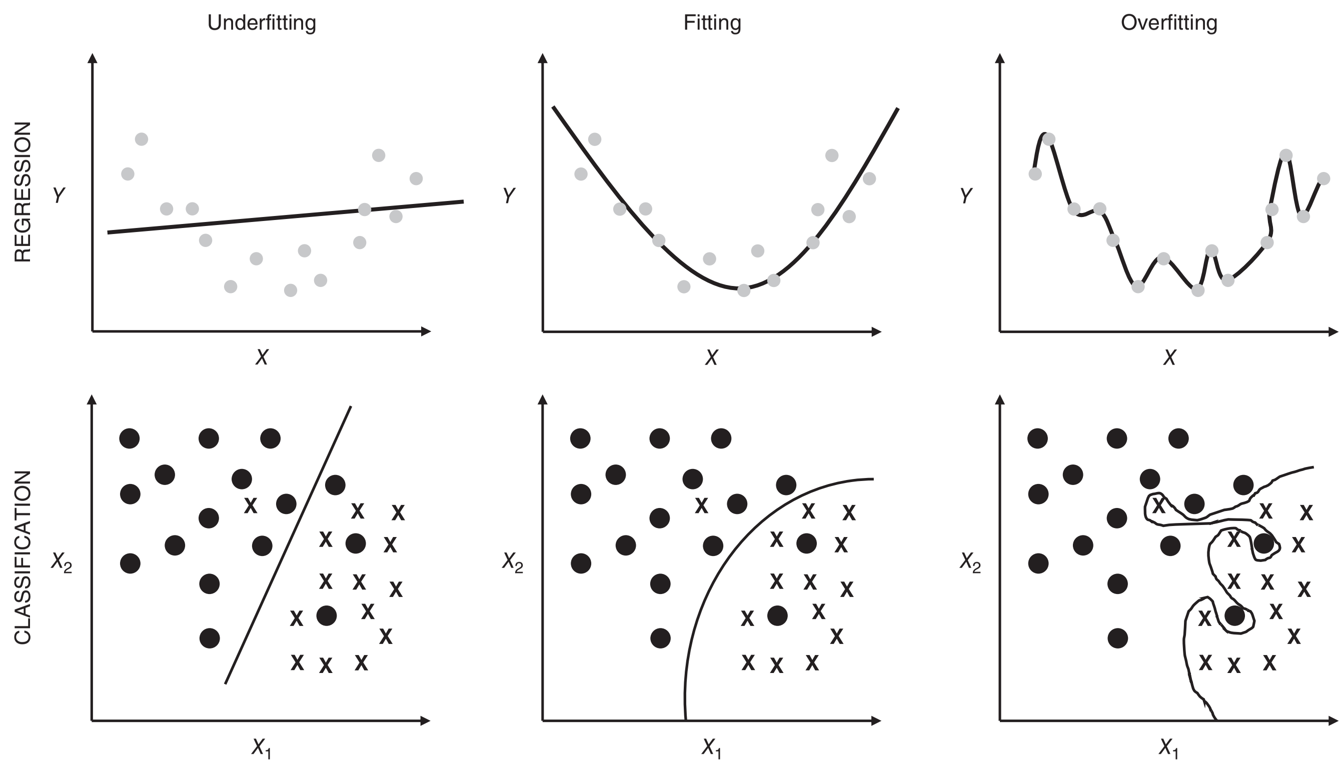

Figure 8.5 Underfitting and overfitting for regression models (top) and for classification models (bottom).

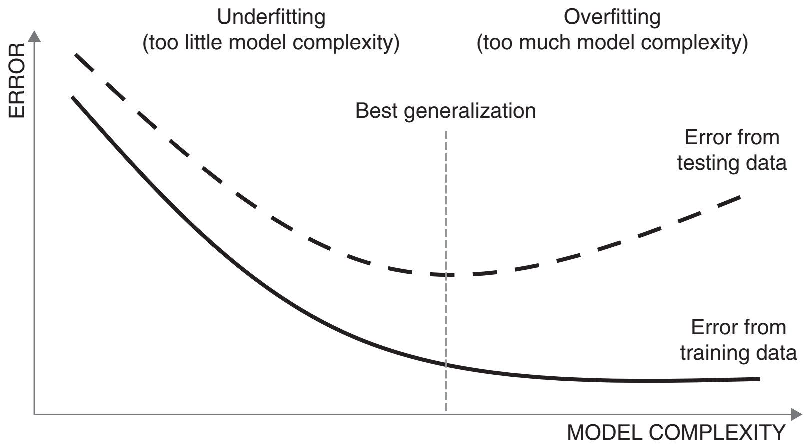

Figure 8.6 Training error versus testing error.

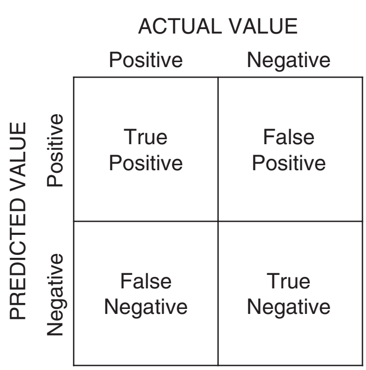

Figure 8.7 The confusion matrix setup.

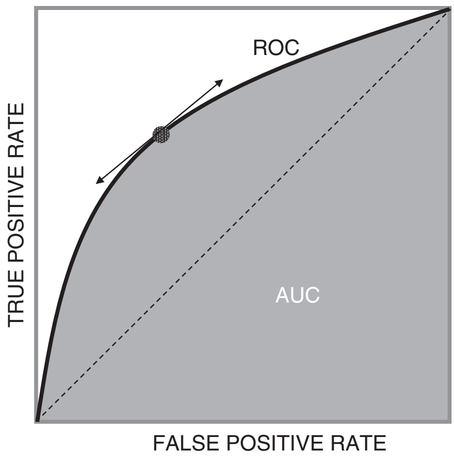

Figure 8.8 Receiver operating characteristics (ROC) curve and the area under the curve (AUC).

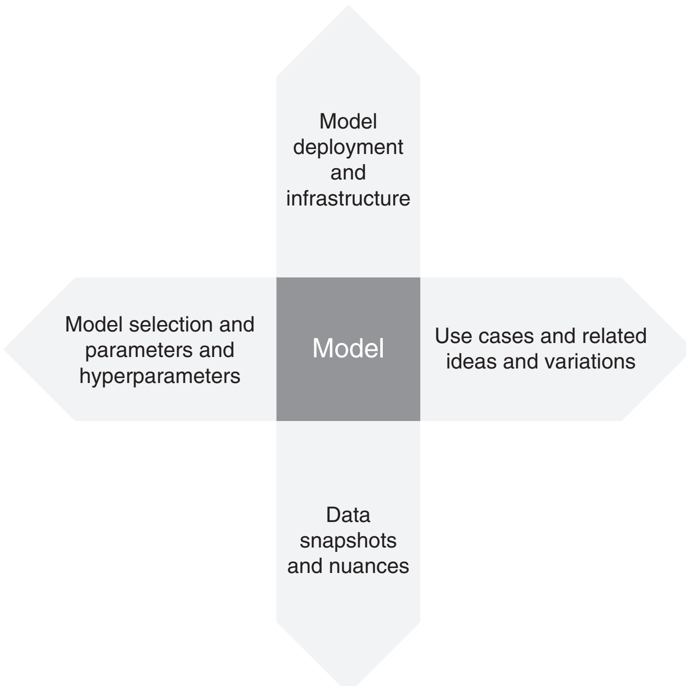

Figure 8.9 Comprehensive model management spans four types of configurations.

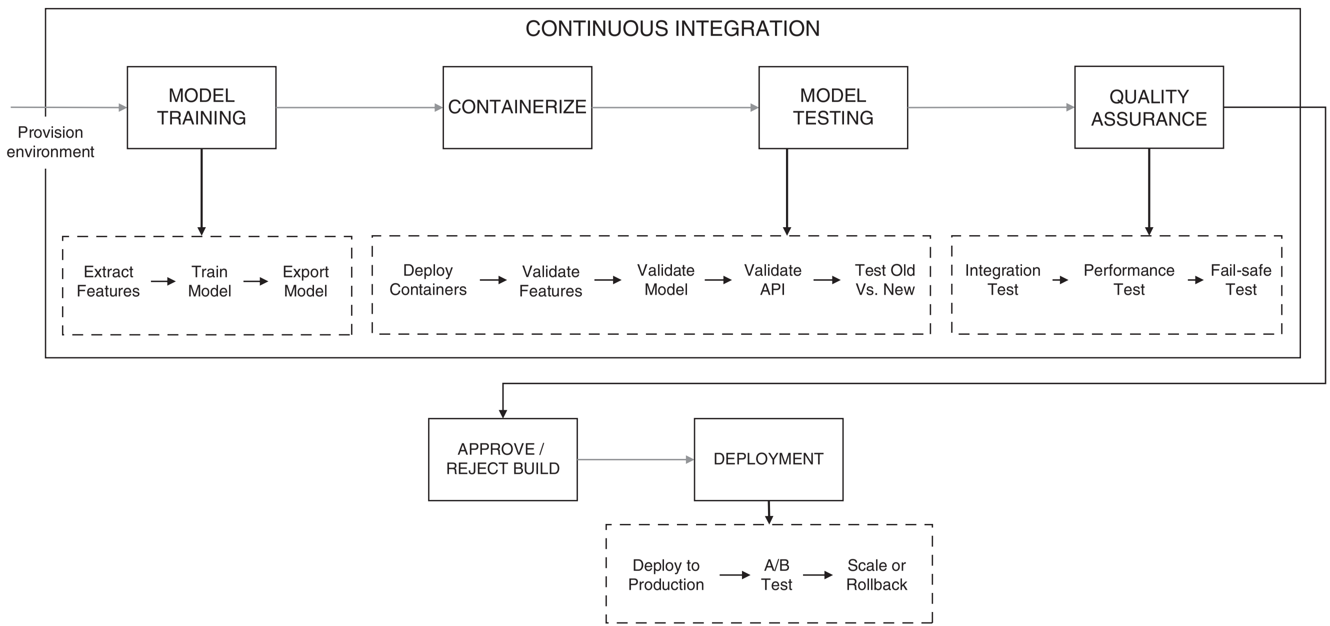

Figure 8.10 AI DevOps process.

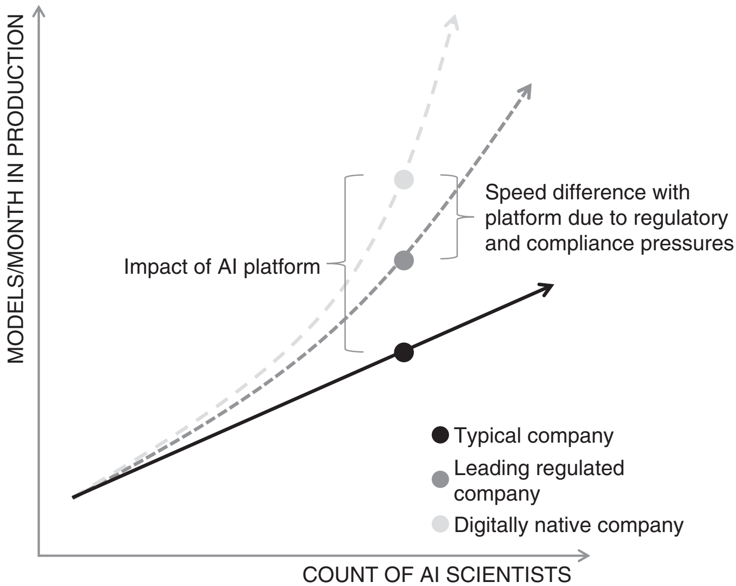

Figure 9.1 Impact of using an AI platform.

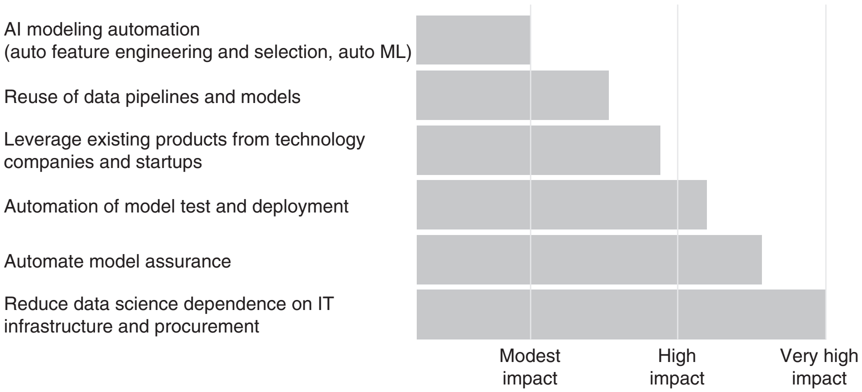

Figure 9.2 Summary of benefits of using an AI platform.

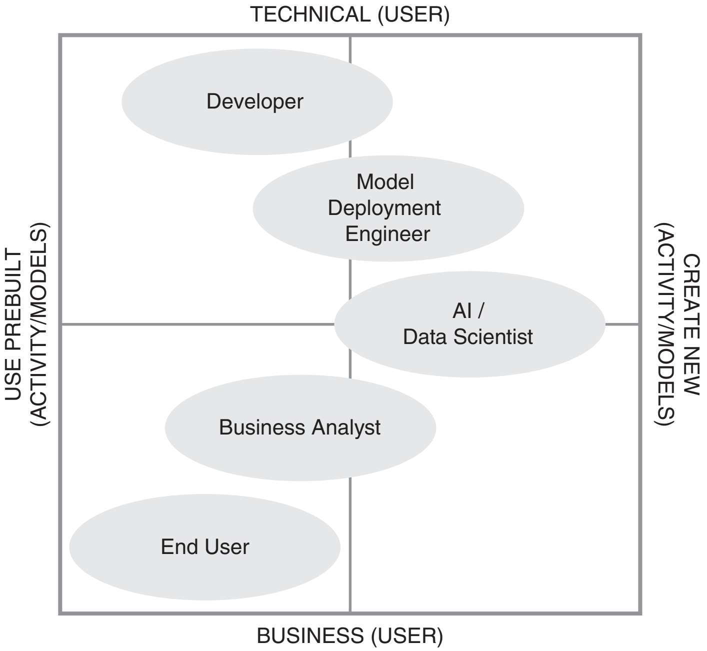

Figure 9.3 Types of users of an AI platform (vertical axis) and how they engage with AI models (horizontal axis).

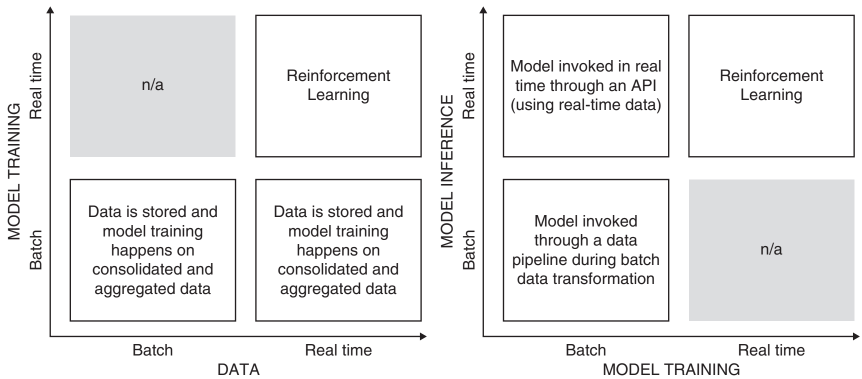

Figure 9.4 Batch versus real time for data, model training, and model inferencing.

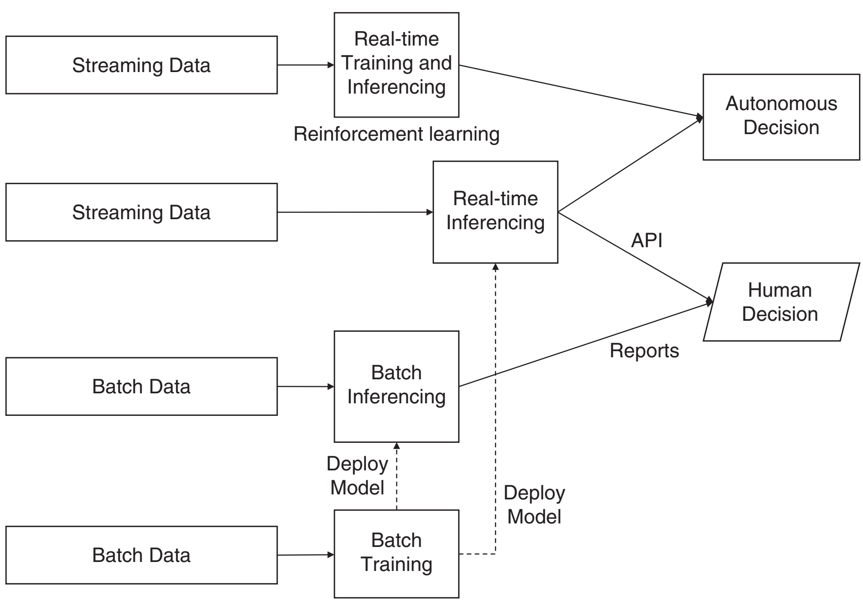

Figure 9.5 The different patterns of batch or streaming data, model training, model inference, and usage.

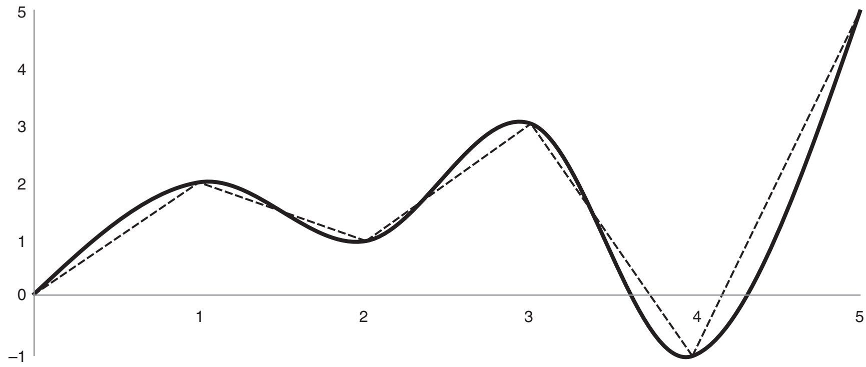

Figure 10.1 Approximating a polynomial function using simpler linear functions in different parts of the *x*-axis.

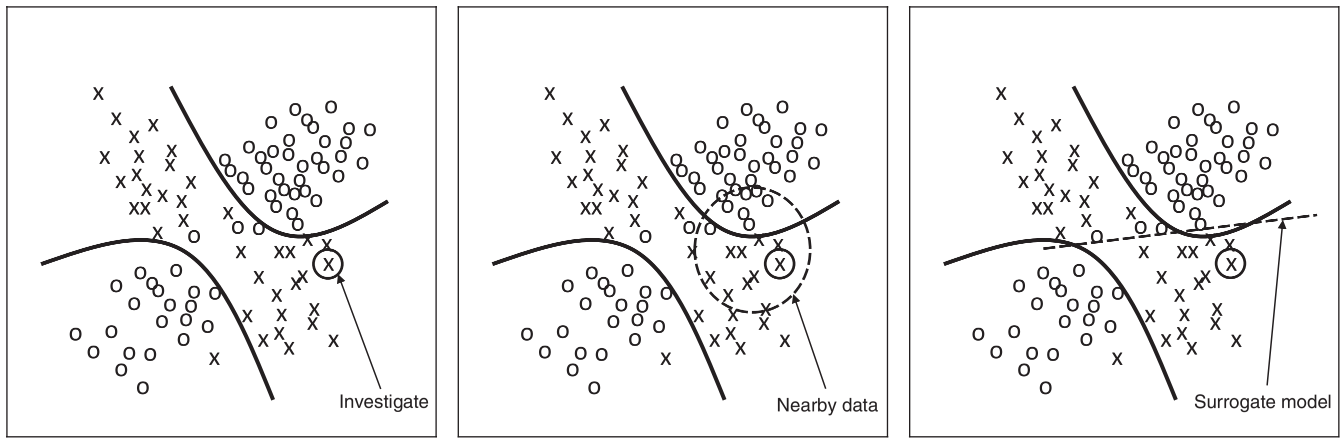

Figure 10.2 An example of how surrogate models can help with interpretability.

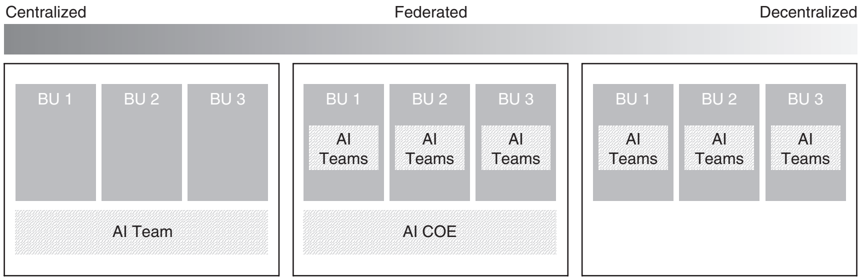

Figure 11.1 Centralized, decentralized, and federated operating models for AI.

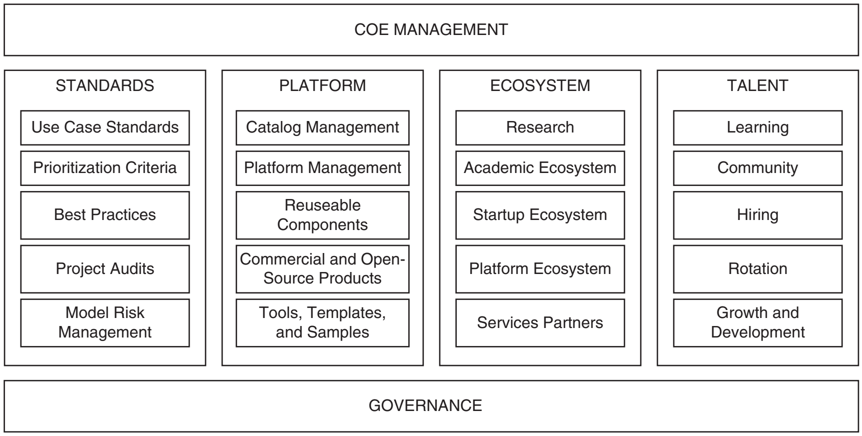

Figure 11.2 Key functions within an AI center of excellence.

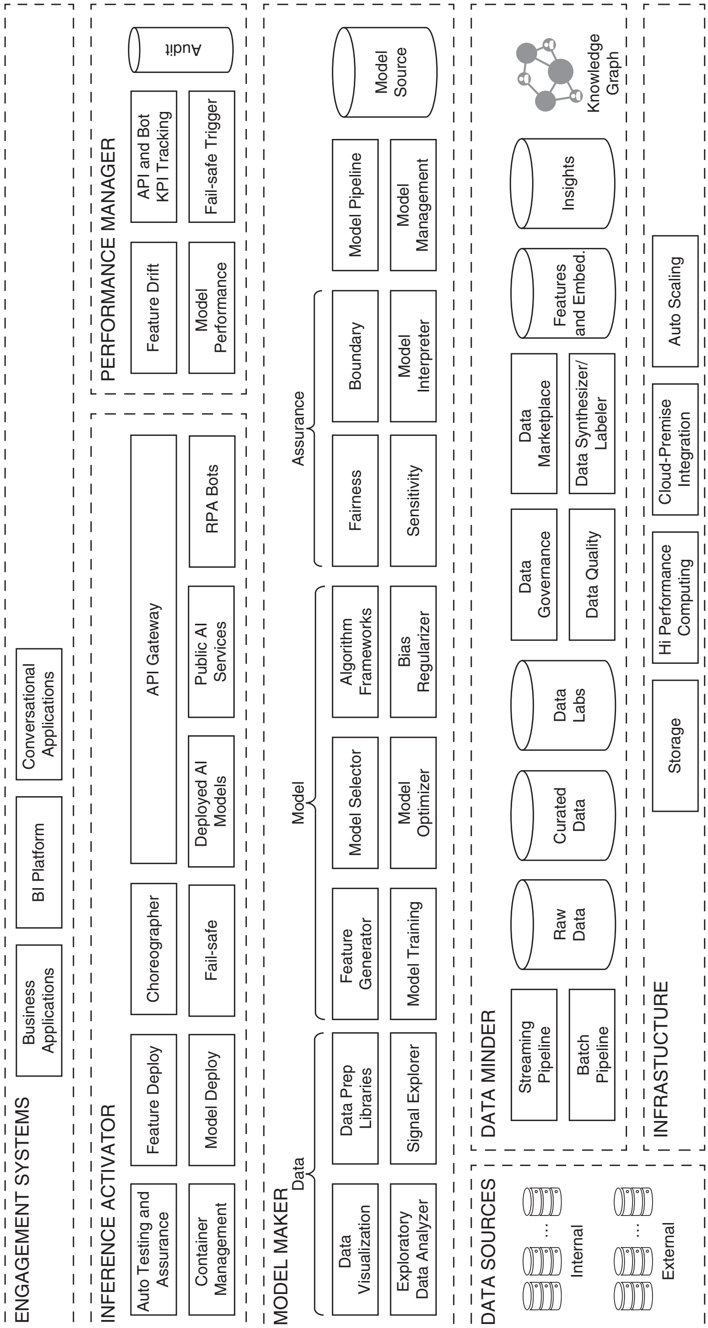

Figure 12.1 Architecture components for an AI platform.

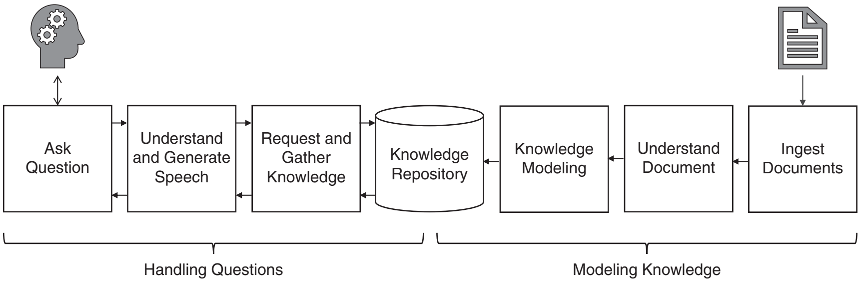

Figure 12.2 Question-and-answer systems built on knowledge modeling.

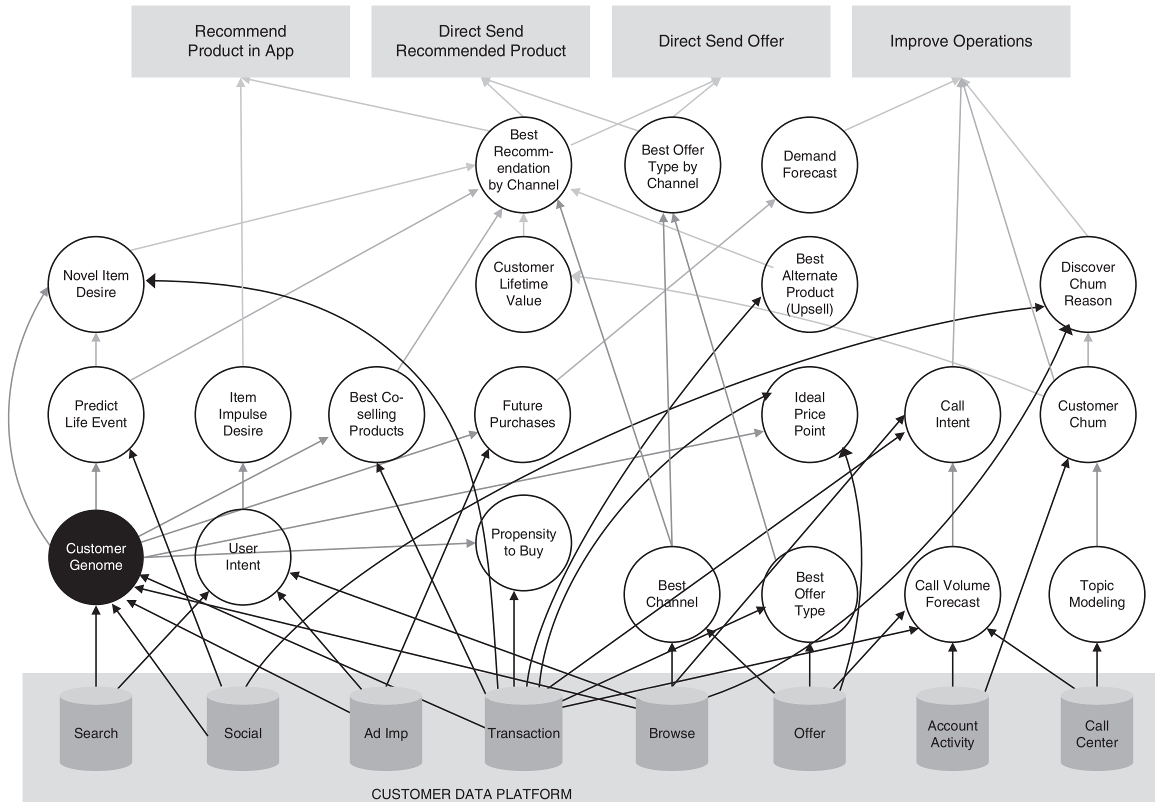

Figure 12.3 Leveraging multiple models for hyperpersonalization.

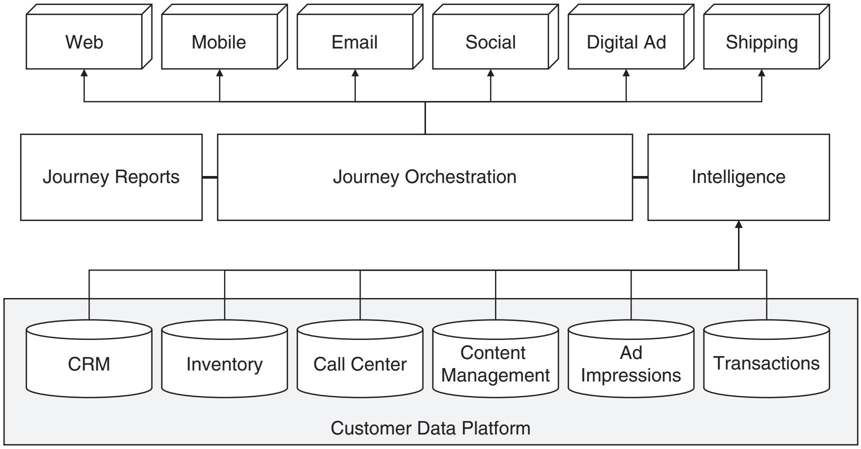

Figure 12.4 Orchestrating personalization interactions.

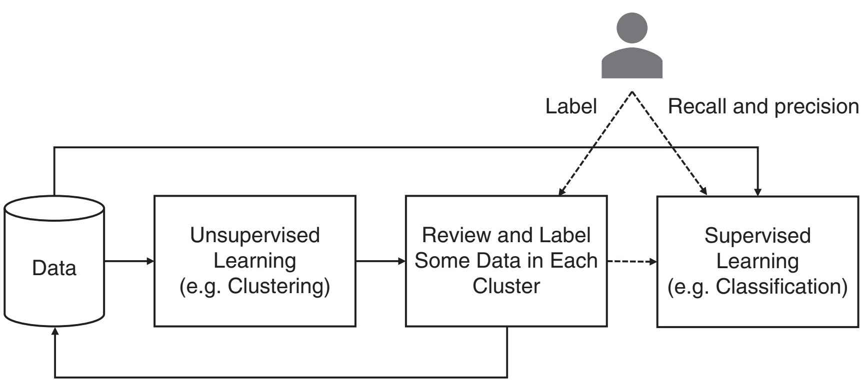

Figure 12.5 Activities for anomaly detection.

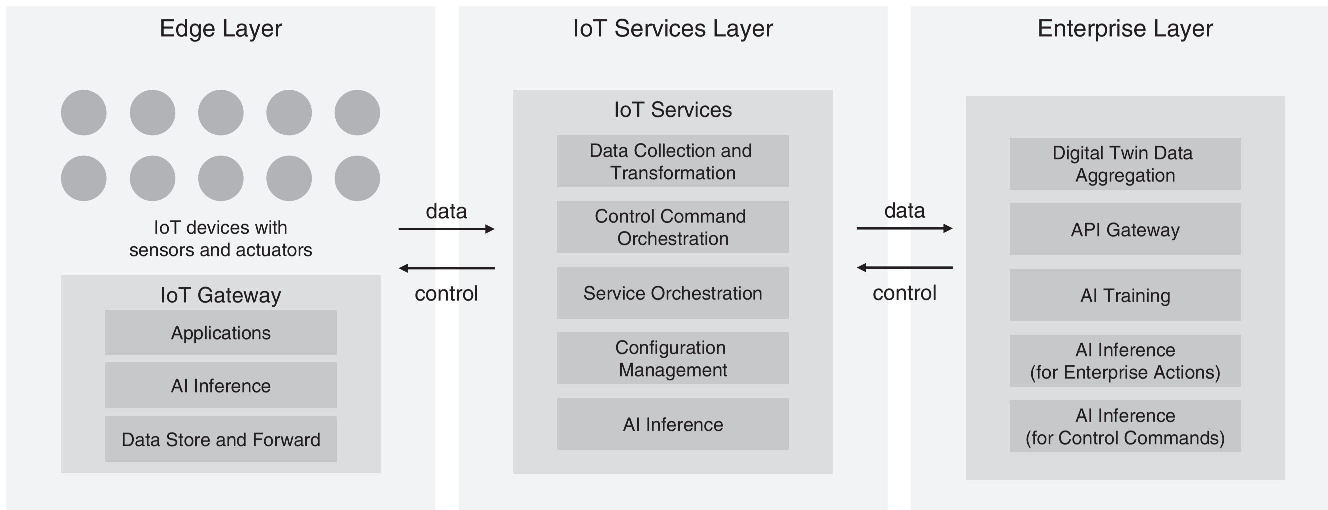

Figure 12.6 Interaction pattern for IoT and edge devices.

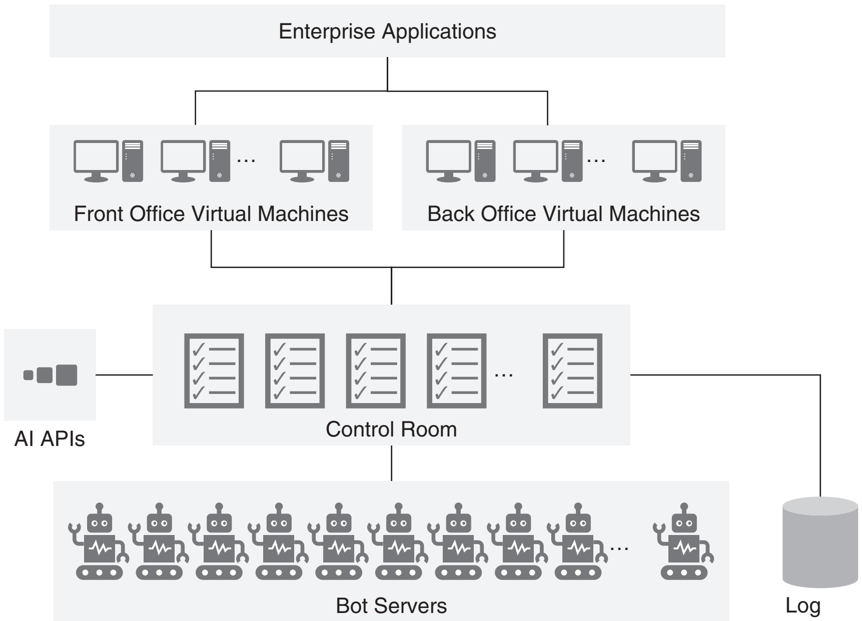

Figure 12.7 RPA-based digital workforce architecture.

```
# import math and data libraries
import pandas as pd
import numpy as np
from scipy.stats import uniform, randint

# import visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D

import missingno as msno

from imblearn.over_sampling import SMOTE

# import sklearn machine learning libraries
from sklearn.preprocessing import LabelBinarizer, label_binarize, Imputer, \
        LabelEncoder, OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

from sklearn.model_selection import train_test_split, cross_val_score, \
        GridSearchCV, KFold, StratifiedKFold, RandomizedSearchCV

# import the necessary model types
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb

# import model performance tools
from sklearn import metrics
from sklearn.metrics import precision_recall_curve, roc_curve, auc, \
        accuracy_score, make_scorer, recall_score, \
        precision_score, confusion_matrix
```

Figure 13.1 Importing relevant libraries that will be used.

```
# set the folder and file names from where you want to get data
folderName = 'gdrive/My Drive/Colab Notebooks/Data/'
fileName = 'customer_churn.csv'

# create dataframe and read file into dataframe
imp_data = pd.read_csv(folderName + fileName)
imp_data.shape
```

```
(3333, 22)
```

Figure 13.2 Importing the data for customer churn.

```
1 imp_data.head()
```

	CUSTOMER_ID	STATE	AREA_CODE	PHONE_NUMBER	ACCOUNT_LENGTH	INTL_PLAN	VMAII
0	10001	KS	415	382-4657	128	no	
1	10002	OH	415	371-7191	107	no	
2	10003	NJ	415	358-1921	137	no	
3	10004	OH	408	375-9999	84	yes	
4	10005	OK	415	330-6626	75	yes	

5 rows × 22 columns

Figure 13.3 Looking at the top few rows of the data.

```
1 #missing data check
2 sns.heatmap(imp_data.isnull(), cbar=False, xticklabels=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e3700c160>
```

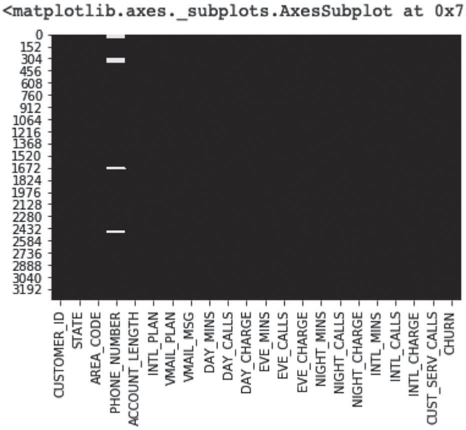

Figure 13.4 Heatmap of missing value. If there were any, they would show as a white bar for that row and column.

```
# drop features that are low impact
proc_data = imp_data.drop(columns=['AREA_CODE','PHONE_NUMBER'])
proc_data.shape

# transforming categorizal data to numerical values
target_features = ['INTL_PLAN', 'VMAIL_PLAN', 'CHURN']
for i, target_feature in enumerate(target_features):
    print(target_feature + " : ", proc_data[target_feature].unique())

# use encoder and transform
encoder = LabelEncoder()
for i, target_feature in enumerate(target_features):
    encoded_values = encoder.fit_transform(proc_data[target_feature].values)
    proc_data[target_feature] = pd.Series(encoded_values, index=imp_data.index)
    # proc_data[target_feature] = proc_data[target_feature].astype('float64')
    print(target_feature + " : ", proc_data[target_feature].unique())
```

```
INTL_PLAN :  ['no' 'yes']
VMAIL_PLAN :  ['yes' 'no']
CHURN :  ['False.' 'True.']
INTL_PLAN :  [0 1]
VMAIL_PLAN :  [1 0]
CHURN :  [0 1]
```

Figure 13.5 Transforming categorical text data to numerical values.

```
# one hot encode categorical values that have more than 2 categories

proc_data = pd.get_dummies(proc_data, columns=['STATE'])
proc_data.shape
```

```
(3333, 71)
```

Figure 13.6 One-hot encoding of US states.

```
# look at distribution of numeric data sets
col_names = ['ACCOUNT_LENGTH','VMAIL_MSG', 'DAY_MINS', 'DAY_CALLS', \
             'DAY_CHARGE', 'EVE_MINS', 'EVE_CALLS', 'EVE_CHARGE', \
             'NIGHT_MINS', 'NIGHT_CALLS', 'NIGHT_CHARGE', 'INTL_MINS', \
             'INTL_CALLS', 'INTL_CHARGE', 'CUST_SERV_CALLS']

fig, axs = plt.subplots(5,3, figsize=(14,17))
for i, col_val in enumerate(col_names):
    sns.distplot(proc_data[col_val], hist=True, ax=axs.flat[i])
    axs.flat[i].set_xlabel(col_val, fontsize=8)
    #axs.flat[i].set_ylabel('Count', fontsize=8)
```

Figure 13.7 Plotting frequency of datasets.

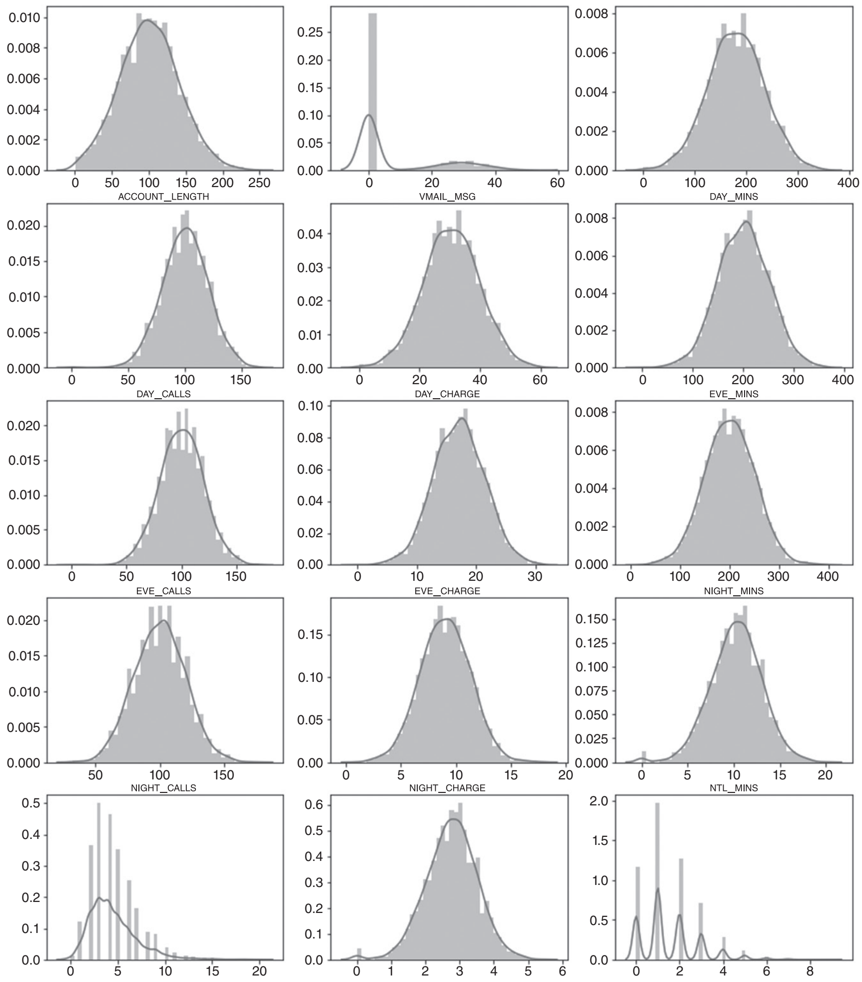

Figure 13.8 Frequency distribution of data of some of the columns.

```
# Create data subset for visualization
states = ['CUSTOMER_ID', 'STATE_AK','STATE_AL','STATE_AR','STATE_AZ', \
          'STATE_CA','STATE_CO','STATE_CT','STATE_DC','STATE_DE','STATE_FL',\
          'STATE_GA','STATE_HI','STATE_IA','STATE_ID','STATE_IL','STATE_IN',\
          'STATE_KS','STATE_KY','STATE_LA','STATE_MA','STATE_MD','STATE_ME',\
          'STATE_MI','STATE_MN','STATE_MO','STATE_MS','STATE_MT','STATE_NC',\
          'STATE_ND','STATE_NE','STATE_NH','STATE_NJ','STATE_NM','STATE_NV',\
          'STATE_NY','STATE_OH','STATE_OK','STATE_OR','STATE_PA','STATE_RI',\
          'STATE_SC','STATE_SD','STATE_TN','STATE_TX','STATE_UT','STATE_VA',\
          'STATE_VT','STATE_WA','STATE_WI','STATE_WV','STATE_WY']
viz_data = proc_data.drop(columns=states)
viz_data.shape

# Compute the correlation matrix
corr = viz_data.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True


# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e37da4198>
```

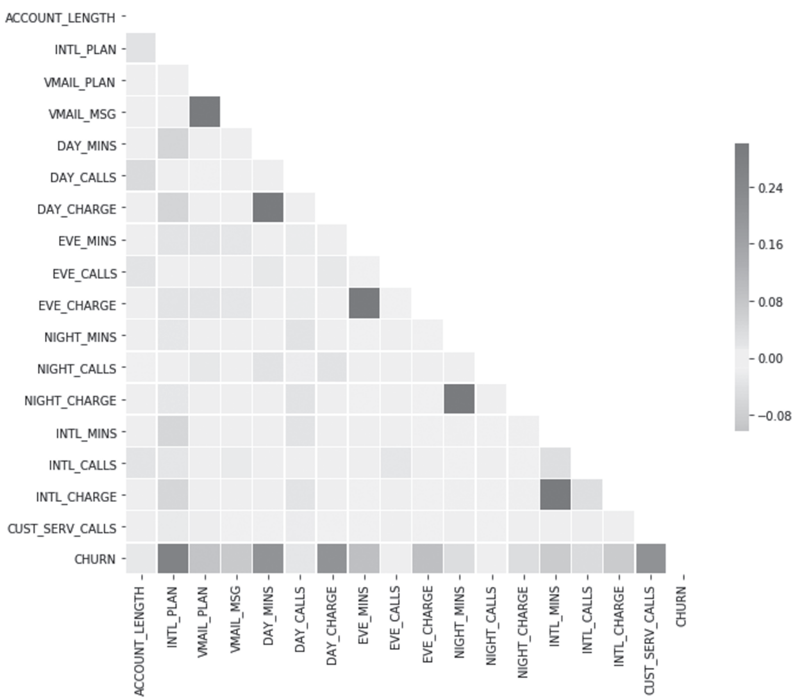

Figure 13.9 Heatmap of the correlations of some of the key columns with each other.

```
# exploring for outliers
col_names = ['ACCOUNT_LENGTH','DAY_MINS', 'DAY_CALLS', 'DAY_CHARGE', \
             'EVE_MINS', 'EVE_CALLS', 'EVE_CHARGE', 'NIGHT_MINS', \
             'NIGHT_CALLS', 'NIGHT_CHARGE']

fig, ax = plt.subplots(1, 10, figsize=(11,5))

for i, col_val in enumerate(col_names):
    sns.boxplot(y=proc_data[col_val], ax=ax[i])
    ax[i].set_ylabel('')
    ax[i].set_xlabel(col_val, fontsize=8)
```

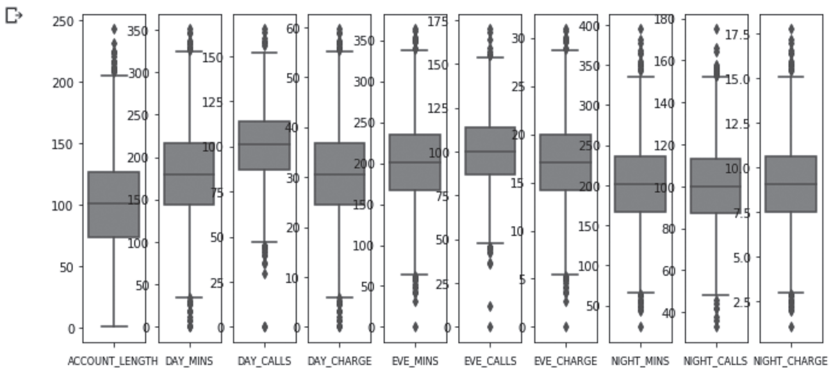

Figure 13.10 Looking for outliers.

```
# look at distribution of categorical data sets
num_col_names = ['INTL_PLAN','VMAIL_PLAN', 'CHURN']


fig, ax =plt.subplots(1, len(num_col_names), figsize=(11,6))
for i, col_val in enumerate(num_col_names):
    sns.countplot(proc_data[col_val], ax=ax[i])
```

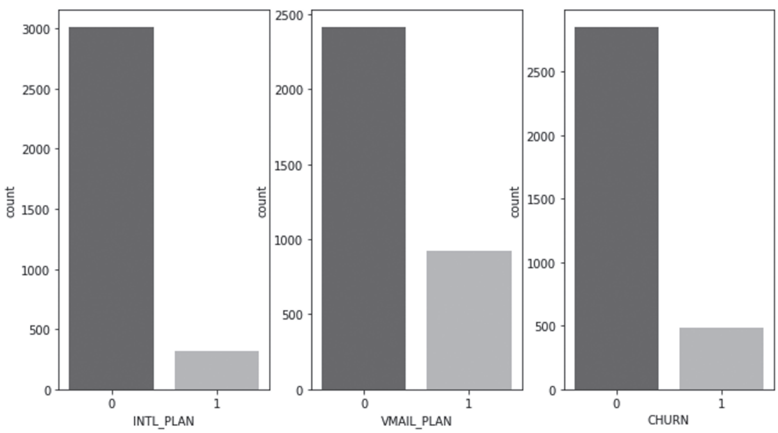

Figure 13.11 Imbalance in label or target data.

```
# scaling the features
scaler = StandardScaler()
scale_cols = ['ACCOUNT_LENGTH','DAY_MINS', 'DAY_CALLS', 'DAY_CHARGE', \
              'EVE_MINS', 'EVE_CALLS', 'EVE_CHARGE', 'NIGHT_MINS', \
              'NIGHT_CALLS', 'NIGHT_CHARGE',
             'VMAIL_MSG', 'INTL_MINS','INTL_CALLS','INTL_CHARGE']
scaled_data = scaler.fit_transform(proc_data[scale_cols])
scaled_data = pd.DataFrame(scaled_data, columns=scale_cols)
scaled_full_data = proc_data.drop(scale_cols, axis=1)
scaled_full_data = pd.concat([scaled_full_data, scaled_data], \
               axis=1, sort=False)
scaled_full_data.shape

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(proc_data['ACCOUNT_LENGTH'], ax=ax1)
sns.kdeplot(proc_data['DAY_MINS'], ax=ax1)
sns.kdeplot(proc_data['DAY_CALLS'], ax=ax1)
sns.kdeplot(proc_data['DAY_CHARGE'], ax=ax1)
sns.kdeplot(proc_data['EVE_MINS'], ax=ax1)
sns.kdeplot(proc_data['EVE_CALLS'], ax=ax1)
sns.kdeplot(proc_data['EVE_CHARGE'], ax=ax1)
sns.kdeplot(proc_data['NIGHT_MINS'], ax=ax1)
sns.kdeplot(proc_data['NIGHT_CALLS'], ax=ax1)
sns.kdeplot(proc_data['NIGHT_CHARGE'], ax=ax1)

ax2.set_title('After Standard Scaler')
sns.kdeplot(scaled_data['ACCOUNT_LENGTH'], ax=ax2)
sns.kdeplot(scaled_data['DAY_MINS'], ax=ax2)
sns.kdeplot(scaled_data['DAY_CALLS'], ax=ax2)
sns.kdeplot(scaled_data['DAY_CHARGE'], ax=ax2)
sns.kdeplot(scaled_data['EVE_MINS'], ax=ax2)
sns.kdeplot(scaled_data['EVE_CALLS'], ax=ax2)
sns.kdeplot(scaled_data['EVE_CHARGE'], ax=ax2)
sns.kdeplot(scaled_data['NIGHT_MINS'], ax=ax2)
sns.kdeplot(scaled_data['NIGHT_CALLS'], ax=ax2)
sns.kdeplot(scaled_data['NIGHT_CHARGE'], ax=ax2)

plt.show()
```

Figure 13.12 Scaling the relevant data columns.

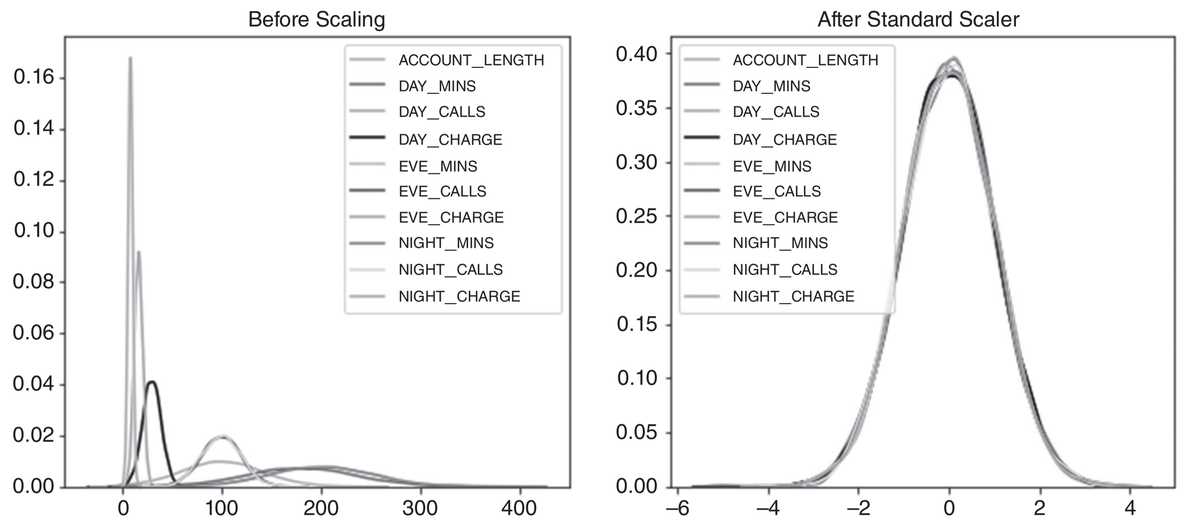

Figure 13.13 Visualizing the data distribution before scaling (left) and after scaling (right).

```python
# remove columns with higher correlations, etc.
scaled_full_data['TOTAL_CHARGE'] = scaled_full_data['DAY_CHARGE'] + \
        scaled_full_data['EVE_CHARGE'] + scaled_full_data['NIGHT_CHARGE'] + \
        scaled_full_data['INTL_CHARGE']
scaled_full_data = scaled_full_data.drop(['DAY_CHARGE', 'EVE_CHARGE', \
        'NIGHT_CHARGE', 'INTL_CHARGE'], axis = 1)
```

Figure 13.14 Dropping individual charge columns and adding the total charge column.

```python
# churn by state, not using the one hot encoding
churn_by_state = pd.crosstab(imp_data.STATE, imp_data.CHURN, normalize='index')
churn_by_state = churn_by_state.sort_values(by='True.')
churn_by_state["True."].plot.bar(title="Churn by State", figsize=(11,3))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e37667208>
```

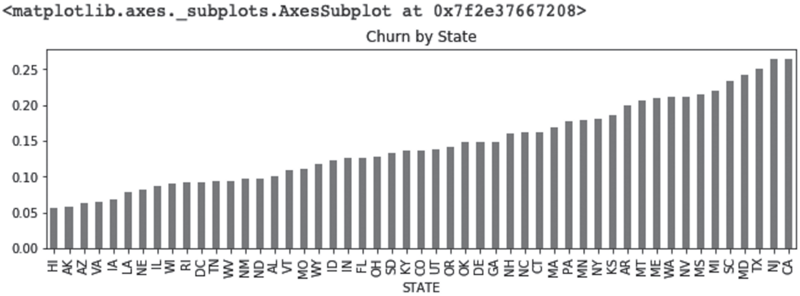

Figure 13.15 Analyzing churn rate by state.

```python
# split the features from the target variable

sourcevars = scaled_full_data.drop(['CHURN', 'CUSTOMER_ID'], axis=1)
targetvar = scaled_full_data['CHURN']

# split the training and validation datasets

xTrain, xTest, yTrain, yTest = train_test_split(sourcevars, targetvar, \
        test_size = 0.25, random_state = 0)

sourcevars.shape, targetvar.shape
```

```
((3333, 65), (3333,))
```

Figure 13.16 Splitting data for training and testing in the ratio of 75:25.

```
# try classification models
model = LogisticRegression(solver = 'lbfgs')

# train the algorithm on training data and predict using the testing data
model.fit(xTrain, yTrain)
predictions = model.predict(xTest)
print("Accuracy : ",accuracy_score(yTest, predictions, normalize = True))
```

```
Accuracy :  0.8477218225419664
```

Figure 13.17 Set up a logistic regression model for binary classification.

```
# this is the accuracy if you assume NO customers will churn
1 - yTest.mean()
```

```
0.8621103117505995
```

Figure 13.18 Percentage of customers that did not churn in the validation dataset.

```
# print(metrics.confusion_matrix(yTest, predictions))
print(pd.DataFrame(confusion_matrix(yTest, predictions),
                   columns=['pred_no_churn', 'pred_churn'],
                   index=['actual_no_churn', 'actual_churn']))
```

```
                 pred_no_churn  pred_churn
actual_no_churn            687          32
actual_churn                95          20
```

```
[173]
# look at performance metrics
print(metrics.classification_report(yTest, predictions))
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       719
           1       0.38      0.17      0.24       115

    accuracy                           0.85       834
   macro avg       0.63      0.56      0.58       834
weighted avg       0.81      0.85      0.82       834
```

Figure 13.19 Looking at the confusion matrix and precision, recall, and F1 score.

```python
# create ROC curve
plt.style.use('ggplot')
y_predict_probabilities = model.predict_proba(xTest)[:,1]

fpr, tpr, _ = roc_curve(yTest, y_predict_probabilities)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2,  \
         label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

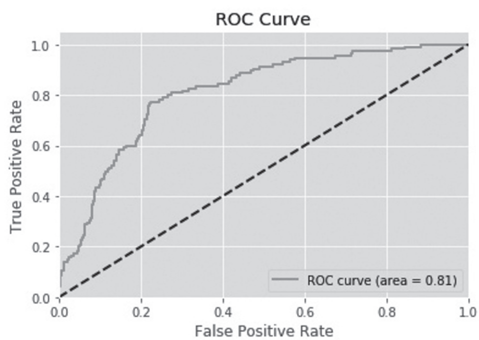

Figure 13.20 Receiver operating characteristic (ROC) curve and area under the curve (AUC).

```
# data augmentation

sm = SMOTE(random_state = 2)
xTrainBal, yTrainBal = sm.fit_sample(xTrain, yTrain.ravel())
predictions = model.fit(xTrainBal, yTrainBal).predict(xTest)
```

[183]

```
print("Accuracy : ",accuracy_score(yTest, predictions, normalize = True))
```

```
Accuracy :  0.7386091127098321
```

[184]

```
# print(metrics.confusion_matrix(yTest, predictions))
print(pd.DataFrame(confusion_matrix(yTest, predictions),
                        columns=['pred_no_churn', 'pred_churn'],
                        index=['actual_no_churn', 'actual_churn']))
```

```
                 pred_no_churn  pred_churn
actual_no_churn            530         189
actual_churn                29          86
```

[185]

```
# look at performance metrics
print(metrics.classification_report(yTest, predictions))
```

```
              precision    recall  f1-score   support

           0       0.95      0.74      0.83       719
           1       0.31      0.75      0.44       115

    accuracy                           0.74       834
   macro avg       0.63      0.74      0.64       834
weighted avg       0.86      0.74      0.78       834
```

Figure 13.21 Augmenting the minority data.

```
1 # try classification models
2 # model = LogisticRegression(solver = 'lbfgs')
3 model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
4
5 # train the algorithm on training data and predict using the testing data
6 model.fit(xTrain, yTrain)
7 predictions = model.predict(xTest)
8 print("Accuracy : ",accuracy_score(yTest, predictions, normalize = True))
```

```
Accuracy :  0.960431654676259
```

```
[189]
# print(metrics.confusion_matrix(yTest, predictions))
print(pd.DataFrame(confusion_matrix(yTest, predictions),
                   columns=['pred_no_churn', 'pred_churn'],
                   index=['actual_no_churn', 'actual_churn']))
```

```
                 pred_no_churn  pred_churn
actual_no_churn            710           9
actual_churn                24          91
```

```
[190]
# look at performance metrics
print(metrics.classification_report(yTest, predictions))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       719
           1       0.91      0.79      0.85       115

    accuracy                           0.96       834
   macro avg       0.94      0.89      0.91       834
weighted avg       0.96      0.96      0.96       834
```

Figure 13.22 Trying a different algorithm – only lines 2 and 3 in the first block have been changed to select a different model.

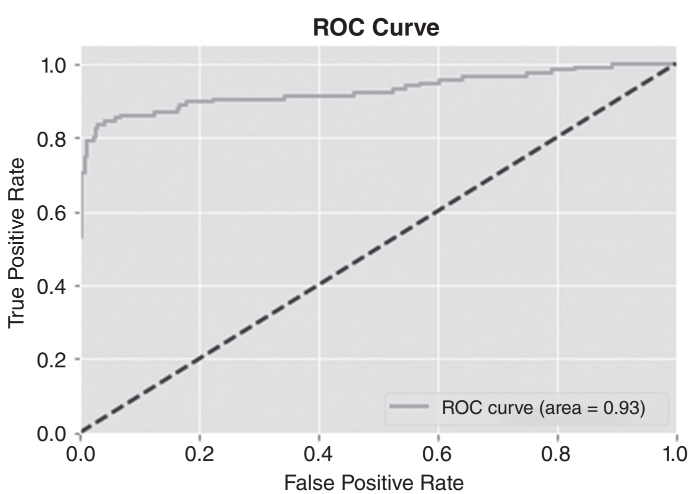

Figure 13.23 ROC curve and AUC using XGBoost.

```
# explore feature importance
fig, ax = plt.subplots(1,1,figsize=(9,6))
xgb.plot_importance(model, max_num_features=10, ax=ax)
plt.show()
```

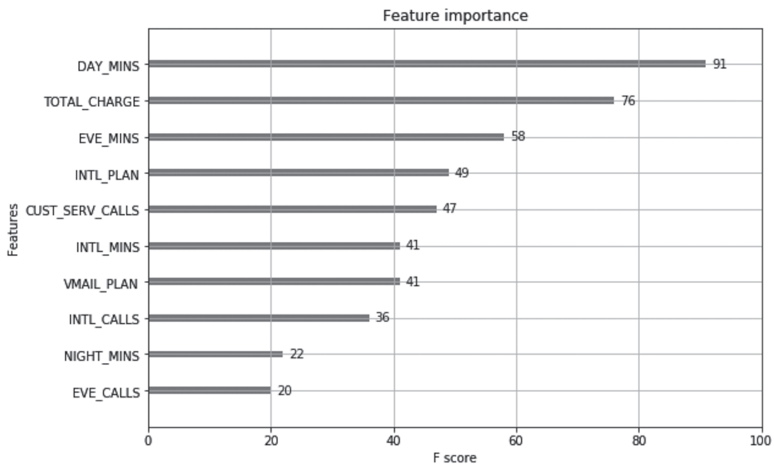

Figure 13.24 Feature importance for the top 10 features in the model.