Contents

1 Data prepping

```
import pandas as pd

# Read file
data = pd.read_csv('/foo/bar.csv', sep=',')

# Group by column to count stuff e.g. how much of dis 'n dat ...
data.groupby(['column1','column2']).count()
data.groupby(['column1','column2'])['column3'].mean().unstack()

# Drop unwanted columns
data = data.drop(['column1','column2'], axis=1)

# Drop rows with null values
data = data.dropna()

# Drop rows with specific value e.g. '?'
data = data[~data.isin(['?']).any(axis=1)]

# Show dimensions and statistical data
data.shape
data.describe()

# One Hot Encoding (if condition, replace with 1, else with 0)
data['column'] = np.where(data['column'] == '...', 0, 1)
data = pd.get_dummies(data, columns=['column'], prefix=['column'])

# Preparing training and test data
import sklearn
from sklearn.model_selection import train_test_split

X = data.drop('column',axis=1)
y = data['column']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30)

# Plotting data
import matplotlib.pyplot as plt

%matplotlib inline

plt.scatter(data['column1'], data['column2'])
plt.xlim([xmin, xmax])
plt.xlabel('column1')
plt.ylabel('column2')
```

2 Classification

2.1 Gaussian Naive Bayes

Prepare the model

```
from sklearn.naive_bayes import GaussionNB

model = GaussianNB()
model.fit(X_train, y_train)
y_predict = model.predict(X_test)
```

Get the accuracy

```
form sklearn.metrics import accuracy_score
accuracy_score(y_test, y_predict)
```

Determine FPrate and FNrate (mostly done after One Hot Encoding)

```
import numpy as np

# In case there are only 2 types
results = pd.Dataframe({'true':y_test, 'estimated':y_predicted})
results['TP'] = np.where((results['true'] == 1) & (results['estimated'] == 1), 1, 0)
results['TN'] = np.where((results['true'] == 0) & (results['estimated'] == 0), 1, 0)
results['FP'] = np.where((results['true'] == 0) & (results['estimated'] == 1), 1, 0)
results['FN'] = np.where((results['true'] == 1) & (results['estimated'] == 0), 1, 0)

FNrate = results['FN'].sum() / (results['FN'].sum() + results['TP'].sum())
FPrate = results['FP'].sum() / (results['FP'].sum() + results['TP'].sum())

# In case there are more types and when One Hot Encoding was not possible
results['TP'] = np.where((results['true'] == results['estimated']), 1, 0)
results['FP'] = np.where((results['true'] != results['estimated']), 1, 0)

FPrate = results['FP'].sum() / (results['FP'].sum() + results['TP'].sum())
```

2.2 Logistic Regression

Prepare the model

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='newton-cg')
model.fit(X_train, y_train)
y_predict = model.predict(X_test)
```

Get the accuracy

```
form sklearn.metrics import accuracy_score
accuracy_score(y_test, y_predict)
```

2.3 Random Forest Classification

Prepare the model

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=...) # n_estimators=100/200/... try different values
model.fit(X_train, y_train)
y_predict = model.predict(X_test)
```

Get the accuracy

```
form sklearn.metrics import accuracy_score
accuracy_score(y_test, y_predict)
```

Get most important features according to the model

```
print(X_train.columns)
print(model.feature_importances_)
pd.Dataframe(model.feature_importances_, \
    columns=['Importance'], \
    index=X_train.columns \
    ).sort_values(by='Importance', ascending=False)
```

Determine FPrate and FNrate (mostly done after One Hot Encoding)

```
import numpy as np

# In case there are only 2 types
results = pd.Dataframe({'true':y_test, 'estimated':y_predicted})
results['TP'] = np.where((results['true'] == 1) & (results['estimated'] == 1), 1, 0)
results['TN'] = np.where((results['true'] == 0) & (results['estimated'] == 0), 1, 0)
results['FP'] = np.where((results['true'] == 0) & (results['estimated'] == 1), 1, 0)
results['FN'] = np.where((results['true'] == 1) & (results['estimated'] == 0), 1, 0)

FNrate = results['FN'].sum() / (results['FN'].sum() + results['TP'].sum())
FPrate = results['FP'].sum() / (results['FP'].sum() + results['TP'].sum())

# In case there are more types and when One Hot Encoding was not possible
results['TP'] = np.where((results['true'] == results['estimated']), 1, 0)
results['FP'] = np.where((results['true'] != results['estimated']), 1, 0)

FPrate = results['FP'].sum() / (results['FP'].sum() + results['TP'].sum())
```

2.4 Ensemble Classification(VotingClassifier)

Prepare the models

```
from sklearn.naive_bayes import GaussionNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, VotingClassifier

gnb = GaussianNB()
rf = RandomForestClassifier(n_estimators=...) # n_estimators=100/200/... try different values
lr = LogisticRegression(solver='newton-g')
model = VotingClassifier(estimators=[('gnb',gnb),('rf',rf),('lr',lr),voting='soft'])

model.fit(X_train, y_train)
y_predict = model.predict(X_test)
```

Get the accuracy

```
form sklearn.metrics import accuracy_score
accuracy_score(y_test, y_predict)
```

3 Regression

3.1 Simple Linear Regression (y = m*x + b)

(1 independent and 1 dependent - e.g. example of petrol cost per milage)

Define round function for plot limits

```
import math

def roundup(x):
    return int(math.ceil(x / 10.0)) * 10

def rounddown(x):
    return int(math.floor(x / 10.0)) * 10

xmin = rounddown(data['column1'].min())
xmax = roundup(data['column1'].max())
```

Look at data

```
import matplotlib.pyplot as plt

%matplotlib inline

plt.scatter(data['column1'], data['column2'])
plt.xlim([xmin, xmax])
plt.xlabel('column1')
plt.ylabel('column2')
```

Prepare the model

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
y_predict = model.predict(X_test)
```

Print the intercept and the slope

```
print(model.intercept_)
print(model.coef_)
```

Plot the training data

```
plt.scatter(X_train, y_train)
plt.xlim([xmin, xmax])
plt.plot(X_train, model.intercept_ + X_train * model.coef_, color='red')
plt.show()
```

Plot the test data

```
plt.scatter(X_test, y_test)
plt.plot(X_test, y_predict, color='red')
plt.xlim([xmin, xmax])
plt.show()
```

Get MAE, MSE, RMSE and R2 *(R2: 0% - 100% -> An R-squared of 100% means that all movements of a security (or another dependent variable) are completely explained by movements in the index)*

```
from sklearn import metrics

MAE = metrics.mean_absolute_error(y_test,y_predict)
```

```
print('Mean Absolute Error: '+ str(MAE))

MSE = metrics.mean_squared_error(y_test,y_predict)
print('Mean Squared Error: '+ str(MSE))

RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_predict))
print('Root Mean Squared Error: '+ str(RMSE))

r2 = metrics.r2_score(y_test,y_predict)
print('R square: ' + str(r2))
```

3.2 Multiple Linear Regression

(1 independent and multiple dependent - e.g example of predicting the price of a house based upon its area, number of bedrooms, the average income of the people in the area, the age of the house, and so on)

Prepare the model

```
from sklearn import metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

model = LinearRegression()
model.fit(X_train,y_train)
y_predict = model.predict(X_test)
```

Print the intercept and the slope

```
print(model.intercept_)
print(model.coef_)
```

Get MAE, MSE, RMSE and R2 *(R2: 0% - 100% -> An R-squared of 100% means that all movements of a security (or another dependent variable) are completely explained by movements in the index)*

```
MAE = metrics.mean_absolute_error(y_test,y_predict)
print('Mean Absolute Error: '+ str(MAE))

MSE = metrics.mean_squared_error(y_test,y_predict)
print('Mean Squared Error: '+ str(MSE))

RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_predict))
print('Root Mean Squared Error: '+ str(RMSE))

r2 = metrics.r2_score(y_test,y_predict)
print('R square: ' + str(r2))
```

3.3 Polynomial Regression

Prepare the model with Linear Regression

```
from sklearn import metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

model = LinearRegression()
model.fit(X_train,y_train)
y_predict = model.predict(X_test)
```

Print the intercept and the slope

```
print(model.intercept_)
print(model.coef_)
```

Get MAE, MSE, RMSE and R2 *(R2: 0% - 100% -> An R-squared of 100% means that all movements of a security (or another dependent variable) are completely explained by movements in the index)*

```
MAE = metrics.mean_absolute_error(y_test,y_predict)
print('Mean Absolute Error: '+ str(MAE))

MSE = metrics.mean_squared_error(y_test,y_predict)
print('Mean Squared Error: '+ str(MSE))

RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_predict))
print('Root Mean Squared Error: '+ str(RMSE))

r2 = metrics.r2_score(y_test,y_predict)
print('R square: ' + str(r2))
```

Prepare the model with Polynomial Regression

```
poly = PolynomialFeatures(degree=8)
X_train_transform = poly.fit_transform(X_train)
X_test_transform = poly.fit_transform(X_test)

model = LinearRegression()
model.fit(X_train_transform,y_train)
y_predict = model.predict(X_test_transform)
```

Print the intercept and the slope

```
print(model.intercept_)
print(model.coef_)
```

Get MAE, MSE, RMSE and R2 *(R2: 0% - 100% -> An R-squared of 100% means that all movements of a security (or another dependent variable) are completely explained by movements in the index)*

```
MAE = metrics.mean_absolute_error(y_test,y_predict)
print('Mean Absolute Error: '+ str(MAE))

MSE = metrics.mean_squared_error(y_test,y_predict)
print('Mean Squared Error: '+ str(MSE))

RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_predict))
print('Root Mean Squared Error: '+ str(RMSE))

mean = data['length'].mean()
print ('Mean: ' + str(mean))
```

```
r2 = metrics.r2_score(y_test,y_predict)
print('R square: ' + str(r2))
```

Calculate the result of the polynomial for a specific value of x

```
def p(x):
    result = model.intercept_
    for i in range(0, len(model.coef_)):
        result += model.coef_[i] * x**i
    return result

# Plot the dataset
plt.scatter(X_test, y_test)
plt.xlim([data['column'].min(), data['column'].max()])
plt.xlabel('age')
plt.ylabel('length')

# Plot the polynomial
t1 = np.arange(1, 6, 0.01)
plt.plot(t1, p(t1), color='red')
plt.show()
```

Loop to create a model for Polynomials of a degree from n to m

```
for i in range(1, 11):
    poly = PolynomialFeatures(degree=i)
    X_train_transform = poly.fit_transform(X_train)
    X_test_transform = poly.fit_transform(X_test)

    model = LinearRegression()
    model.fit(X_train_transform,y_train)

    y_predict = model.predict(X_test_transform)

    RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_predict))
    print('Root Mean Squared Error for i = ' + str(i) + ' is ' + str(RMSE))
    print()
```

3.4 Random Forest Regression

Prepare the model

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100) # n_estimators=100/200/... try different values
model.fit(X_train,y_train)
y_predict = model.predict(X_test)
```

Create a dictionary and loop over the different models

```
# create dictionary of models
modeldict = {'modelForest25':RandomForestRegressor(n_estimators=25),\
             'modelForest50':RandomForestRegressor(n_estimators=50),\
             'modelForest100':RandomForestRegressor(n_estimators=100),\
             'modelForest150':RandomForestRegressor(n_estimators=150),\
             'modelForest200':RandomForestRegressor(n_estimators=200),\
             'modelForest250':RandomForestRegressor(n_estimators=250)}

# initialize MAE by choosing a high value
MAE = 100000

# initialize bestmodel
bestmodel = 'modelForest25'

for modelkey in modeldict:
    model = modeldict[modelkey]

    model.fit(X_train,y_train)

    y_predict = model.predict(X_test)

    NEWMAE = mean_absolute_error(y_test,y_predict)
    if NEWMAE < MAE:
        MAE = NEWMAE
        bestmodel = modelkey

print('Bestmodel: ' + modelkey)
print('Mean Absolute Error: '+ str(MAE))
r2 = r2_score(y_test,y_predict)
print('R square: ' + str(r2))
```

4 Document Classification

4.1 Natural Language Processing (NLP)

Detect language

```
import pandas as pd
import numpy as np
import nltk
from langdetect import detect

detect("When the sun goes down")
# OR when working with a table
data['Lang'] = data['column1'].apply(detect)
```

Create a pivot table

```
pd.pivot_table(data, values='column1', index=['column2'],columns=['Lang'], aggfunc='count').fillna(0)
```

Tokenization

```
# importing word_tokenize from nltk
from nltk.tokenize import word_tokenize

# download punkt otherwise you get
# an error: Resource 'tokenizers/punkt/english.pickle' not found.
nltk.download('punkt')

# sample text for performing tokenization
text = "In Brazil they drive on the right-hand side of the road."

# Passing the string text into word tokenize for splitting the text into tokens.
token = word_tokenize(text)
print(token)

# finding the frequency distinct in the tokens
# Importing FreqDist library from nltk and passing token into FreqDist
from nltk.probability import FreqDist
fdist = FreqDist(token)
print(fdist)

# To find the frequency of top 10 words
fdist1 = fdist.most_common(10)
print(fdist1)
```

Remove stopwords

```
# Stopwords
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string

def remove_stopwords_en(text):
    stop_words_en = set(stopwords.words('english'))
    punctuations="?:!.,;<>/\+-"
    words = word_tokenize(text)
    result = ''
    for word in words:
```

```
        if word not in stop_words_en and word not in punctuations:
            result += word + ' '
    return result

text = "Cristiano Ronaldo was born on February 5, 1985, in Funchal, Madeira, Portugal."
print(remove_stopwords_en(text))
# OR when working with a table
data['column1'] = data['column1'].map(remove_stopwords_en)
```

Stemming

```
# Stemming: examples
from nltk.stem.snowball import SnowballStemmer

# english
englishStemmer=SnowballStemmer("english")

stm = ["welcome", "welcoming"]
for word in stm:
    print(word + ":" + englishStemmer.stem(word))
print()

stm = ["ball", "balls"]
for word in stm:
    print(word + ":" + englishStemmer.stem(word))
print()

stm = ["waited", "waiting", "waits"]
for word in stm:
    print(word + ":" + englishStemmer.stem(word))
print()

stm = ["giving", "give", "given", "gave"]
for word in stm:
    print(word + ":" + englishStemmer.stem(word))
print()

# dutch
dutchStemmer=SnowballStemmer("dutch")

stm = ["worden", "wordt"]
for word in stm:
    print(word + ":" + dutchStemmer.stem(word))
print()

stm = ["dader", "daders", "daad"]
for word in stm:
    print(word + ":" + dutchStemmer.stem(word))
print()

stm = ["las", "lezen", "gelezen", "lees"]
for word in stm:
    print(word + ":" + dutchStemmer.stem(word))
print()

# Stemming: replace words by stem
from nltk.stem.snowball import SnowballStemmer
```

```
def stemming_en(text):
    englishStemmer=SnowballStemmer("english")
    words = word_tokenize(text)
    result = ''
    for word in words:
        result += englishStemmer.stem(word) + ' '
    return result


text = "That Arizona sky burning in your eyes. \
        You look at me and, babe, I wanna catch on fire. \
        It's buried in my soul like California gold. \
        You found the light in me that I couldn't find."
print(stemming_en(text))
# OR when working with a table
data['column1'] = data['column1'].map(remove_stopwords_en)
```

Lemmatization

```
# Lemmatization
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

print("rocks:", lemmatizer.lemmatize("rocks"))
print("corpora:", lemmatizer.lemmatize("corpora"))

words = ["gone", "going", "went"]
for word in words:
    print(word + ":" + lemmatizer.lemmatize(word))
```

4.2 Example: email classification (Multinomial Naive Bayes and others included)

Prepare data

```
import pandas as pd
import numpy as np
import nltk

pd.set_option('display.max_colwidth', 200)  # set width of column output

ads = pd.read_csv('input/emailADS.CSV',encoding = "ISO-8859-1")
ict = pd.read_csv('input/emailICT.CSV',encoding = "ISO-8859-1")
job = pd.read_csv('input/emailJOB.CSV',encoding = "ISO-8859-1")
# news = pd.read_csv('input/emailnews.csv',encoding = "ISO-8859-1")

# Merge data into dataset email
email = ads.append([ict,job])

# Keep the columns Subject, Body and Category
email = email[['Subject','Body','Category']]
email.head()

# Combine Subject and Body into a single field "Text"
email["Text"] = email['Subject'] + ' ' + email['Body']

# Drop the columns Subject and Body afterwards
email = email.drop(['Subject','Body'], axis=1)
email.head()
```

Language detection - this may take a while!

```
from langdetect import detect
email['Lang'] = email['Text'].apply(detect)
```

Create a pivot table that shows the number of emails per category (row) and per lang (column)

```
pd.pivot_table(email, \
               values='Text', \
               index=['Category'], \
               columns=['Lang'], \
               aggfunc='count').fillna(0)
```

Create emailnl which only contains the emails with Lang = nl

```
emailnl = email[email['Lang'] == 'nl']
emailnl.head()
```

Remove the stopwords

```
# Indeed a number of Dutch e-mails are incorrectly labeled as "English"
# We decide to create a model for the dutch e-mails only.
# Stopwords
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string

def remove_stopwords_nl(text):
    stop_words_nl = set(stopwords.words('dutch'))
    punctuations="?:!.,;<>/\+-"
    words = word_tokenize(text)
```

```
    result = ''
    for word in words:
        if word not in stop_words_nl and word not in punctuations:
            result += word + ' '
    return result

email['Text'] = email['Text'].map(remove_stopwords_nl)
email.head()
```

Stemming: replace words by stem

```
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer

def stemming_en(text):
    englishStemmer=SnowballStemmer("dutch")
    words = word_tokenize(text)
    result = ''
    for word in words:
        result += englishStemmer.stem(word) + ' '
    return result

email['Text'] = email['Text'].map(stemming_en)
email.head()
```

Prepare model

```
from sklearn.model_selection import train_test_split
X = emailnl.drop('Category',axis=1)
y = emailnl['Category']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30)

# Apply TfidVectorizer = Give the number of different words
# When data not already split into test and train data, use the normal
#   dataframe => X = vec.fit_transform(data['Text'])
from sklearn.feature_extraction.text import TfidfVectorizer

vec = TfidfVectorizer()
X = vec.fit_transform(X_train['Text'])
print(len(vec.get_feature_names()))

# Use ensemble classification
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.pipeline import make_pipeline # Needed for MultinomialNB

lr = LogisticRegression(solver='newton-cg')
rf = RandomForestClassifier(n_estimators=150)
nb =  MultinomialNB() # Needed for MultinomialNB

voting = VotingClassifier(estimators=[('lr', lr), ('rf', rf), ('mnb', nb)], voting='hard')
# Needed for MultinomialNB(in case there is only nb, replace voting with nb)
model = make_pipeline(TfidfVectorizer(), voting)

model.fit(X_train["Text"], y_train)
categories = model.predict(X_test["Text"])
```

Get the accuracy

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, categories)*100)
```

Prediction function

```
def predict_category(s, model=model):
    s = remove_stopwords_nl(s)
    s = stemming_en(s)
    pred = model.predict([s])
    return pred[0]

predict_category("Alles aan halve prijs en nog veel goedkoper!")
```

5 Neural Networks

5.1 General stuff - Image classification

Prepare test and training data

```
import numpy as np

# keras import for the dataset
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Normalize the data

```
# let's print the shape before we reshape and normalize
print("X_train shape", X_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test.shape)
print("y_test shape", y_test.shape)

# building the input vector from the 28x28 pixels
# = linearize the image to get a 784 (= 28x28) vector
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)

# normalizing the data to help with the training
# normalized data leads to better models
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

# print the final input shape ready for training
print("Train matrix shape", X_train.shape)
print("Test matrix shape", X_test.shape)
```

5.2 Image classification with a Random Forest Classifier

Prepare the model

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=300)
model.fit(X_train, y_train)
y_test2 = model.predict(X_test)
```

Get the accuracy

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_test2)
```

5.3 Image classification with Artificial Neural Networks

Normalize the data

```
# let's print the shape before we reshape and normalize
print("X_train shape", X_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test.shape)
print("y_test shape", y_test.shape)

# building the input vector from the 28x28 pixels = linearize the image to get a 784 (= 28x28) vector
```

```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)

# normalizing the data to help with the training
# normalized data leads to better models
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

# print the final input shape ready for training
print("Train matrix shape", X_train.shape)
print("Test matrix shape", X_test.shape)
```

One-Hot Encoding with numpy series

```
# one-hot encoding using keras' numpy-related utilities
from tensorflow.keras.utils import to_categorical

y_train = to_categorical(y_train)
print(y_train.shape)
print(y_train[0])  # one sample's categorical data
y_test = to_categorical(y_test)
print(y_test.shape)
```

Use TfidfVectorizer to transform X

```
# Apply TfidVectorizer = Give the number of different words
from sklearn.feature_extraction.text import TfidfVectorizer

vec = TfidfVectorizer()
X = vec.fit_transform(data['sentence'])
print(len(vec.get_feature_names()))

# If error on compiling model, create the test and training data after TfidVectorizer
# X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30)
```

Build neural network (a linear stack of layers with the sequential model)

```
# keras imports for building our neural network
from keras.models import Sequential, load_model
from keras.layers.core import Dense, Dropout, Activation

# building a linear stack of layers with the sequential model
model = Sequential()
model.add(Dense(512, input_shape=(784,))) # = len(vec.get_feature_names())
model.add(Activation('sigmoid'))
model.add(Dropout(0.2))

model.add(Dense(512))
model.add(Activation('sigmoid'))
model.add(Dropout(0.2))

model.add(Dense(10)) # The amount of classes that the output layer has
model.add(Activation('softmax'))
```

Compiling the sequential model

```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
```

Training the model and saving metrics in history

```
model.fit(X_train, y_train,epochs=20,verbose=2)
```

Saving the model

```
import os
save_dir = "./"
model_name = 'keras_mnist.h5'
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
```

Use the saved model for calculating it's predictive accuracy

```
mnist_model = load_model('keras_mnist.h5')
loss, accuracy = mnist_model.evaluate(X_test, y_test)

print("Test Loss", loss)
print("Test Accuracy", accuracy)
```

Load the model and create predictions on the test set

```
model = load_model('keras_mnist.h5')

predictions = model.predict(X_test)
print(y_test[0])

# Check the probabilities returned by predict for first test sample
for index, probability in enumerate(predictions[0]):
    print(f'{index}: {probability:.10%}')
```

Locating the Incorrect Predictions

```
incorrect_predictions = []

for i, (p, e) in enumerate(zip(predictions, y_test)):
    predicted, expected = np.argmax(p), np.argmax(e)

    if predicted != expected:  # prediction was incorrect
        incorrect_predictions.append((i, predicted, expected))
```

5.4 Convolutional Neural Network

Adding a convolution layer

```
from tensorflow.keras.models import Sequential
cnn = Sequential()

from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D
cnn.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
```

Adding a pooling layer

```
cnn.add(MaxPooling2D(pool_size=(2, 2)))
```

Adding Another Convolutional Layer and Pooling Layer

```
cnn.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
```

Add a dense layer to reduce the number of features

```
cnn.add(Dense(units=128, activation='relu'))
```

Adding another dense layer to produce the final output

```
cnn.add(Dense(units=10, activation='softmax'))
```

Printing the Model's Summary

```
cnn.summary()
```

Visualizing a Model's Structure

```
from tensorflow.keras.utils import plot_model
from IPython.display import Image
plot_model(cnn, to_file='convnet.png', show_shapes=True, show_layer_names=True)
Image(filename='convnet.png')  # display resulting image in notebook
```

Get the accuracy

```
loss, accuracy = cnn.evaluate(X_test, y_test)
print(loss)
print(accuracy)
```