Building a Smarter AI-Powered Spam Classifier

**Phase 4 Submission Document**

**Project Title** : Development Part 2

# **Topic:** section begin building your project by loading and preprocessing the dataset



**INTRODUCTION:**

In the realm of data-driven projects, success often hinges on the quality and readiness of the dataset under examination. Loading and preprocessing this data is a foundational step, setting the stage for robust analysis, modeling, and decision-making. In this section, we will delve into the critical processes of acquiring, loading, and preparing the dataset for our project.

|  |  |
| --- | --- |
| **Dataset Overview:** | We will begin by providing a brief overview of the dataset under |



investigation. This includes its source, the context in which it was collected, and the primary objective of its utilization within the project.

|  |  |
| --- | --- |
| **Data Acquisition:** | This section will discuss the methods employed to obtain the dataset. |



It may include data collection procedures, sources, and any ethical considerations associated with data gathering.

|  |  |
| --- | --- |
| **Data Loading:** | Loading the dataset into our analysis environment is a pivotal task. We |
| will disc | uss the tools and techniques used for importing the data, whether it be from a |



database, CSV file, API, or other sources.

|  |  |
| --- | --- |
| **Data Preprocessing:** | Raw data seldom arrives in the perfect format for analysis. This |



subsection will cover data preprocessing steps such as handling missing values, dealing with outliers, and converting data types to ensure it is ready for analytical tasks.

|  |  |
| --- | --- |
| **Data Exploration:** | While primarily an exploratory process, this phase is crucial in |



identifying initial patterns and trends within the data, which may inform subsequent project directions.

|  |  |
| --- | --- |
| **Data Quality Assurance:** | Quality control is integral to ensuring the integrity of the |



dataset. We will discuss measures taken to validate and clean the data, maintaining its accuracy and reliability.

**DATASET :**



**Context:**

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged acording being ham (legitimate) or spam.

**Content:**

The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.

This corpus has been collected from free or free for research sources at the Internet:

* A collection of 425 SMS spam messages was manually extracted from the Grumbletext Web site. This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the very spam message received. The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involved carefully scanning hundreds of web pages.



A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available.

* A list of 450 SMS ham messages collected from Caroline Tag's PhD Thesis available.

* Finally, we have incorporated the SMS Spam Corpus v.0.1 Big. It has 1,002 SMS ham messages and 322 spam messages and it is public available.

* This is an automatically-generated kernel with starter code demonstrating how to read in the data and begin exploring. Click the blue "Edit Notebook" or "Fork Notebook" button at the top of this kernel to begin editing.

**Acknowledgements:**

The original dataset can be found [here.](https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection) The creators would like to note that in case you find the dataset useful, please make a reference to previous paper and the web page:<http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/>in your papers, research, etc. We offer a comprehensive study of this corpus in the following paper. This work presents a number of statistics, studies and baseline results for several machine learning methods.

**Exploratory Analysis:**

To begin this exploratory analysis, first use matplotlib to import libraries and define functions for plotting the data. Depending on the data, not all plots will be made. (Hey, I'm just a kerneling bot, not a Kaggle Competitions Grandmaster!)

ln[1]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| from mpl\_toolkits.mplot3d import Axes3D from sklearn.preprocessing import StandardScaler import matplotlib.pyplot as plt *# plotting* import numpy as np *# linear algebra* import os *# accessing directory structure*  import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)* | | | | |
| **There is 1 csv file in the current version of the dataset**: | | | /input'): | |
|  | | |
| ln[2]: | dirname, \_, filenames **in** os.walk('/kaggle for filename **in** filenames: print(os.path.join(dirname, filename)) | |
| for |
| The next hidden code cells define functions for plotting data. Click on the "Code" button in the publis | | | | , |
| hed kernel to reveal the hidden code. | | *graphs (histogram/bar graph) of column data* df, nGraphShown, nGraphPerRow):  if nunique[col] > 1 **and** nunique[col] < 50]] *# For have between 1 and 50 unique values*  nGraphPerRow - 1) / nGraphPerRow figsize = (6 \* nGraphPerRow, 8 \* nGraphRow), dpi = 80  'k') nGraphShown)): nGraphPerRow, i + 1) i] type(columnDf.iloc[0]), np.number)):  columnDf.value\_counts() bar()  90)  **]}** (column **{i}**)') , w\_pad = 1.0, h\_pad = 1.0) graphWidth): | |
|  | |
| ln[3]: | *# Distribution* plotPerColumnDistribution( nunique = df.nunique() df = df[[col for col **in** df  *displaying purposes, pick columns that* nRow, nCol = df.shape columnNames = list(df) nGraphRow = (nCol + plt.figure(num = None,  facecolor = 'w', edgecolor = for i **in** range(min(nCol,  plt.subplot(nGraphRow, columnDf = df.iloc[:, if (**not** np.issubdtype( valueCounts = valueCounts.plot. else: columnDf.hist() plt.ylabel('counts') plt.xticks(rotation = plt.title(f'**{columnNames[i** plt.tight\_layout(pad = 1.0 plt.show()    *# Correlation matrix* plotCorrelationMatrix(df, filename = df.dataframeName |
| def                  ln[4]:  def |

|  |  |
| --- | --- |
| df = df.dropna('columns') *# drop columns with NaN*  df = df[[col for col **in** df if df[col].nunique() > 1]] *# keep columns where th ere are more than 1 unique values* if df.shape[1] < 2: print(f'No correlation plots shown: The number of non-NaN or constant col umns (**{df.shape[1]}**) is less than 2') return corr = df.corr()  plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgecolor='k')  corrMat = plt.matshow(corr, fignum = 1)  plt.xticks(range(len(corr.columns)), corr.columns, rotation=90) plt.yticks(range(len(corr.columns)), corr.columns) plt.gca().xaxis.tick\_bottom() plt.colorbar(corrMat)  plt.title(f'Correlation Matrix for **{filename}**', fontsize=15) plt.show()  ln[5]:    *# Scatter and density plots* def plotScatterMatrix(df, plotSize, textSize): df = df.select\_dtypes(include =[np.number]) *# keep only numerical columns*  *# Remove rows and columns that would lead to df being singular* df = df.dropna('columns')  df = df[[col for col **in** df if df[col].nunique() > 1]] *# keep columns where th ere are more than 1 unique values* columnNames = list(df)  if len(columnNames) > 10: *# reduce the number of columns for matrix inversion of kernel density plots*  columnNames = columnNames[:10] df = df[columnNames]  ax = pd.plotting.scatter\_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')  corrs = df.corr().values for i, j **in** zip(\*plt.np.triu\_indices\_from(ax, k = 1)): ax[i, j].annotate('Corr. coef = **%.3f**' % corrs[i, j], (0.8, 0.2), xycoords  ='axes fraction', ha='center', va='center', size=textSize) plt.suptitle('Scatter and Density Plot') plt.show() | |
| **Now you're ready to read in the data and use the plotting functions to visualize the data.** | ) |
| ln[6]:  nRowsRead = 1000 *# specify 'None' if want to read whole file*  *# spam.csv has 5572 rows in reality, but we are only loading/previewing the first*  *1000 rows*  df1 = pd.read\_csv('/kaggle/input/spam.csv', delimiter=',', nrows = nRowsRead df1.dataframeName = 'spam.csv' nRow, nCol = df1.shape  print(f'There are **{nRow}** rows and **{nCol}** columns') |
|  | |

--------------------------------------------------------------------------- UnicodeDecodeError Traceback (most recent call last) pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_convert\_tokens()

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_convert\_with\_dtype()

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_string\_convert()

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.\_string\_box\_utf8()

UnicodeDecodeError: 'utf-8' codec can't decode bytes in position 135-136: invalid continuation byte

**During handling of the above exception, another exception occurred:**

UnicodeDecodeError Traceback (most recent call last)

<ipython-input-6-556be88e201a> in <module>

1. nRowsRead = 1000 # specify 'None' if want to read whole file
2. # spam.csv has 5572 rows in reality, but we are only loading/previewing t he first 1000 rows

----> 3 df1 = pd.read\_csv('/kaggle/input/spam.csv', delimiter=',', nrows = nRowsR ead)

1. df1.dataframeName = 'spam.csv'
2. nRow, nCol = df1.shape

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in parser\_f(filepath\_ or\_buffer, sep, delimiter, header, names, index\_col, usecols, squeeze, prefix, ma ngle\_dupe\_cols, dtype, engine, converters, true\_values, false\_values, skipinitial space, skiprows, skipfooter, nrows, na\_values, keep\_default\_na, na\_filter, verbos e, skip\_blank\_lines, parse\_dates, infer\_datetime\_format, keep\_date\_col, date\_pars er, dayfirst, cache\_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, d ialect, error\_bad\_lines, warn\_bad\_lines, delim\_whitespace, low\_memory, memory\_map

, float\_precision)

**683** )

**684**

--> 685 return \_read(filepath\_or\_buffer, kwds)

**686**

**687** parser\_f.\_\_name\_\_ = name

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in \_read(filepath\_or\_ buffer, kwds)

**461**

**462** try:

--> 463 data = parser.read(nrows)

1. finally:
2. parser.close()

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in read(self, nrows)

1. def read(self, nrows=None):
2. nrows = \_validate\_integer("nrows", nrows)

-> 1154 ret = self.\_engine.read(nrows)

**1155**

**1156** # May alter columns / col\_dict

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in read(self, nrows)

1. def read(self, nrows=None):
2. try:

-> 2048 data = self.\_reader.read(nrows)

1. except StopIteration:
2. if self.\_first\_chunk:

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.read()

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_read\_low\_memory()

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_read\_rows()

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_convert\_column\_data( )

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_convert\_tokens()

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_convert\_with\_dtype()

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.TextReader.\_string\_convert()

pandas/\_libs/parsers.pyx in pandas.\_libs.parsers.\_string\_box\_utf8()

UnicodeDecodeError: 'utf-8' codec can't decode bytes in position 135-136: invalid continuation byte

**Let's take a quick look at what the data looks like:**

|  |
| --- |
| ln[7]:  df1.head(5) |

---------------------------------------------------------------------------

NameError Traceback (most recent call last)

<ipython-input-7-e55bb665ba13> in <module>

----> 1 df1.head(5)

NameError: name 'df1' is not defined

**Distribution graphs (histogram/bar graph) of sampled columns:**

|  |
| --- |
| ln[8]:  plotPerColumnDistribution(df1, 10, 5) |

--------------------------------------------------------------------------- NameError Traceback (most recent call last)

<ipython-input-8-a0a199b2d778> in <module>

----> 1 plotPerColumnDistribution(df1, 10, 5)

NameError: name 'df1' is not defined

|  |
| --- |
| import pandas as pd  from sklearn.preprocessing import StandardScaler  from sklearn.impute import SimpleImputer  from sklearn.compose import ColumnTransformer  from sklearn.pipeline import Pipeline  from sklearn.preprocessing import OneHotEncoder  # Load the dataset from a CSV file  dataset = pd.read\_csv("dataset.csv")  # Display the first few rows of the dataset  print("Initial dataset:")  print(dataset.head())  # Define preprocessing steps  numeric\_features = ['numerical\_feature1', 'numerical\_feature2']  categorical\_features = ['categorical\_feature']  # Create transformers for numeric and categorical data  numeric\_transformer = Pipeline(steps=[  ('imputer', SimpleImputer(strategy='mean')),  ('scaler', StandardScaler())  ])  categorical\_transformer = Pipeline(steps=[  ('imputer', SimpleImputer(strategy='most\_frequent')),  ('onehot', OneHotEncoder(handle\_unknown='ignore'))  ])  # Combine transformers using ColumnTransformer  preprocessor = ColumnTransformer(  transformers=[  ('num', numeric\_transformer, numeric\_features),  ('cat', categorical\_transformer, categorical\_features)  ])  # Create a pipeline for the entire preprocessing process  pipeline = Pipeline(steps=[('preprocessor', preprocessor)])  # Apply preprocessing to the dataset  preprocessed\_data = pipeline.fit\_transform(dataset)  # Display the preprocessed dataset  print("\nPreprocessed dataset:")  preprocessed\_df = pd.DataFrame(preprocessed\_data)  print(preprocessed\_df.head()) |

|  |
| --- |
|  |
|  |
|  |
|  |

**Conclusion:**

This concludes your starter analysis! To go forward from here, click the blue "Edit Notebook" button at the top of the kernel. This will create a copy of the code and environment for you to edit. Delete, modify, and add code as you please. Happy Kaggling!