Tally Solutions Pvt. Ltd.

Invoice Date Extraction Model Using Tesseract OCR Engine and Tensorflow

WORKFLOW :

READ A FOLDER CONTAINING PDF INVOICE DOCUMENTS

CONVERT THESE PDFS TO JPEG AND STORE IN ANOTHER FOLDER

NOW PREPROCESS THESE IMAGES FOR BETTER OCR EXTRACTION

APPLY TESSERACT OCR TO EACH IMAGE IN DATASET

STORE THE OUTPUT IN DATAFRAME FORMAT

MODIFY THIS TESSERACT OUTPUT FOR BETTER RESULTS

PROCESS THE TEXT GENERATED AND FIND ALL DATE FIELDS

GET NEIGHBOUR INFORMATION FOR EACH DATE FIELD

NORMALIZE THE DATAFRAME

NOW CREATE A DATASET (.CSV) FILE OUT OF THIS DATAFRAME

ANNOTATE THIS DATAFRAME FOR TRUE INVOICE DATE

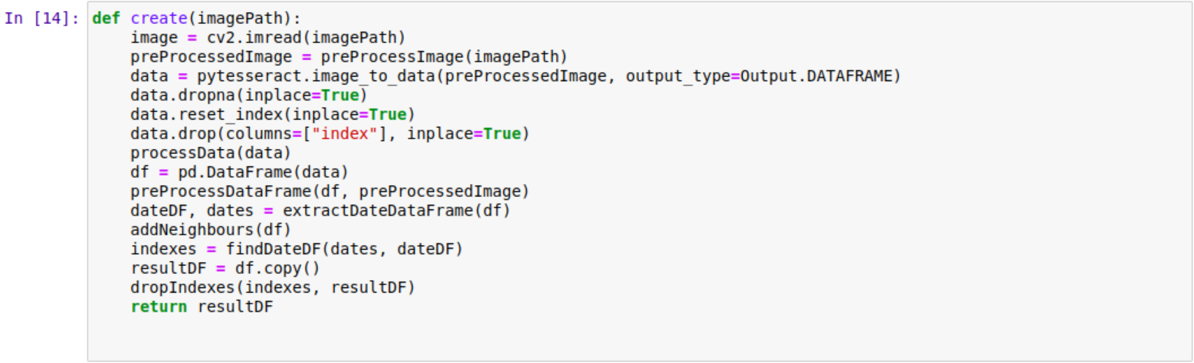
READ THIS CSV AND ADD ZONAL INFO FOR FIELDS

PREPROCESS THE DATASET FOR MODEL TRAINING

PREPARE / TRAIN / TEST / SAVE A KERAS MODEL

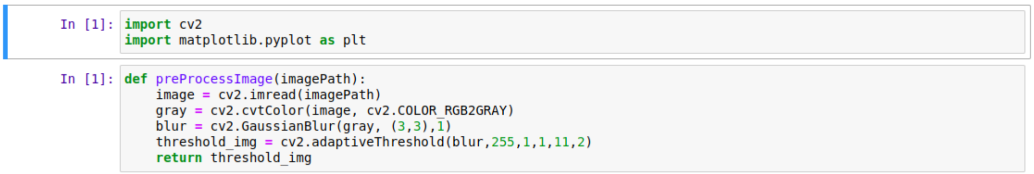
USE THIS MODEL FOR INFERENCE

DATASET PREPARATION :



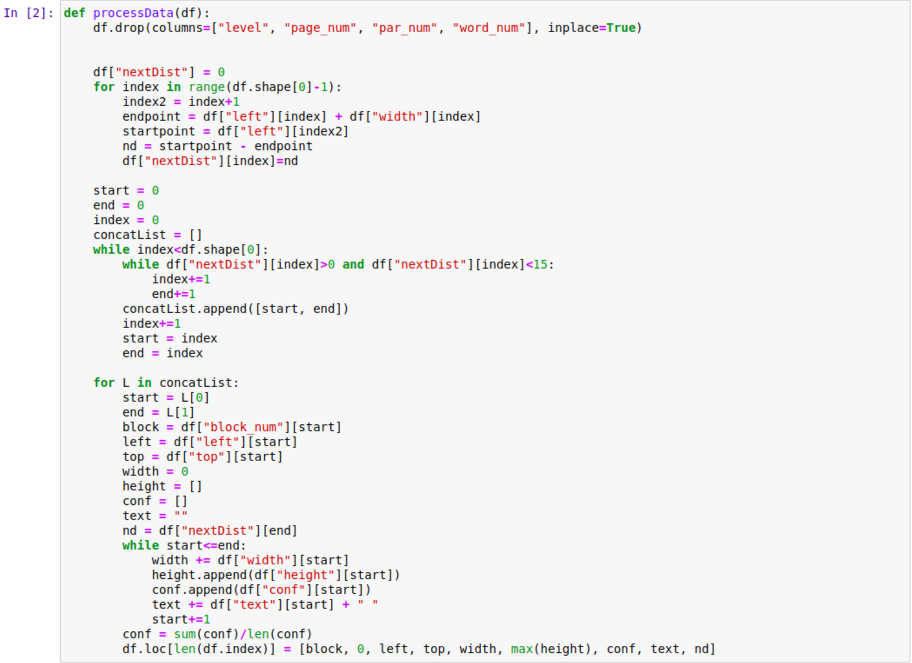
```
In [14]: def create(imagePath):
             image = cv2.imread(imagePath)
             preProcessedImage = preProcessImage(imagePath)
             data = pytesseract.image_to_data(preProcessedImage, output_type=Output.DATAFRAME)
             data.dropna(inplace=True)
             data.reset_index(inplace=True)
             data.drop(columns=["index"], inplace=True)
             processData(data)
             df = pd.DataFrame(data)
             preProcessDataFrame(df, preProcessedImage)
             dateDF, dates = extractDateDataFrame(df)
             addNeighbours(df)
             indexes = findDateDF(dates, dateDF)
             resultDF = df.copy()
             dropIndexes(indexes, resultDF)
             return resultDF
```

- Read the image
- Preprocessed the image



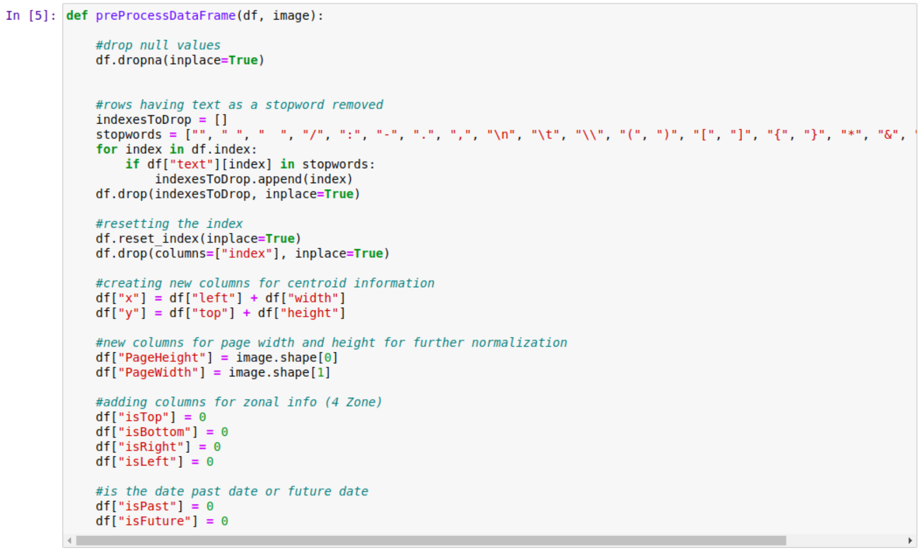
```
In [1]: import cv2
        import matplotlib.pyplot as plt

In [1]: def preProcessImage(imagePath):
            image = cv2.imread(imagePath)
            gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            blur = cv2.GaussianBlur(gray, (3,3),1)
            threshold_img = cv2.adaptiveThreshold(blur,255,1,1,11,2)
            return threshold_img
```

- Applied tesseract OCR on the image
- Modified tesseract output to concat seperated tokens in a single word



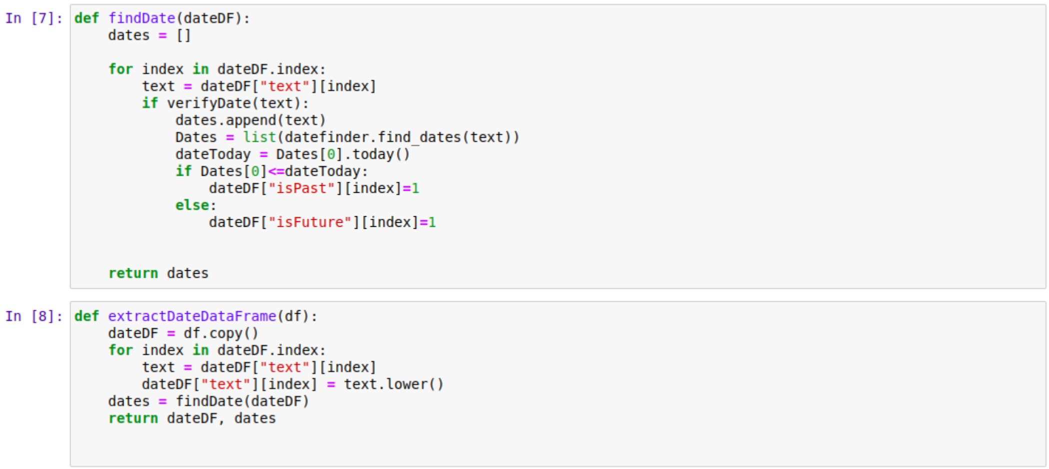
```
In [2]: def processData(df):
            df.drop(columns=["level", "page_num", "par_num", "word_num"], inplace=True)


            df["nextDist"] = 0
            for index in range(df.shape[0]-1):
                index2 = index+1
                endpoint = df["left"][index] + df["width"][index]
                startpoint = df["left"][index2]
                nd = startpoint - endpoint
                df["nextDist"][index]=nd

            start = 0
            end = 0
            index = 0
            concatList = []
            while index<df.shape[0]:
                while df["nextDist"][index]>0 and df["nextDist"][index]<15:
                    index+=1
                    end+=1
                concatList.append([start, end])
                index+=1
                start = index
                end = index

            for L in concatList:
                start = L[0]
                end = L[1]
                block = df["block_num"][start]
                left = df["left"][start]
                top = df["top"][start]
                width = 0
                height = []
                conf = []
                text = ""
                nd = df["nextDist"][end]
                while start<=end:
                    width += df["width"][start]
                    height.append(df["height"][start])
                    conf.append(df["conf"][start])
                    text += df["text"][start] + " "
                    start+=1
                conf = sum(conf)/len(conf)
                df.loc[len(df.index)] = [block, 0, left, top, width, max(height), conf, text, nd]
```

- PreProcessed the extracted dataframe from tesseract output



```
In [5]: def preProcessDataFrame(df, image):

            #drop null values
            df.dropna(inplace=True)


            #rows having text as a stopword removed
            indexesToDrop = []
            stopwords = ["", " ", "  ", "/", ":", "-", ".", ",", "\n", "\t", "\\", "(", ")", "[", "]", "{", "}", "*", "&", '
            for index in df.index:
                if df["text"][index] in stopwords:
                    indexesToDrop.append(index)
            df.drop(indexesToDrop, inplace=True)

            #resetting the index
            df.reset_index(inplace=True)
            df.drop(columns=["index"], inplace=True)

            #creating new columns for centroid information
            df["x"] = df["left"] + df["width"]
            df["y"] = df["top"] + df["height"]

            #new columns for page width and height for further normalization
            df["PageHeight"] = image.shape[0]
            df["PageWidth"] = image.shape[1]

            #adding columns for zonal info (4 Zone)
            df["isTop"] = 0
            df["isBottom"] = 0
            df["isRight"] = 0
            df["isLeft"] = 0

            #is the date past date or future date
            df["isPast"] = 0
            df["isFuture"] = 0
```

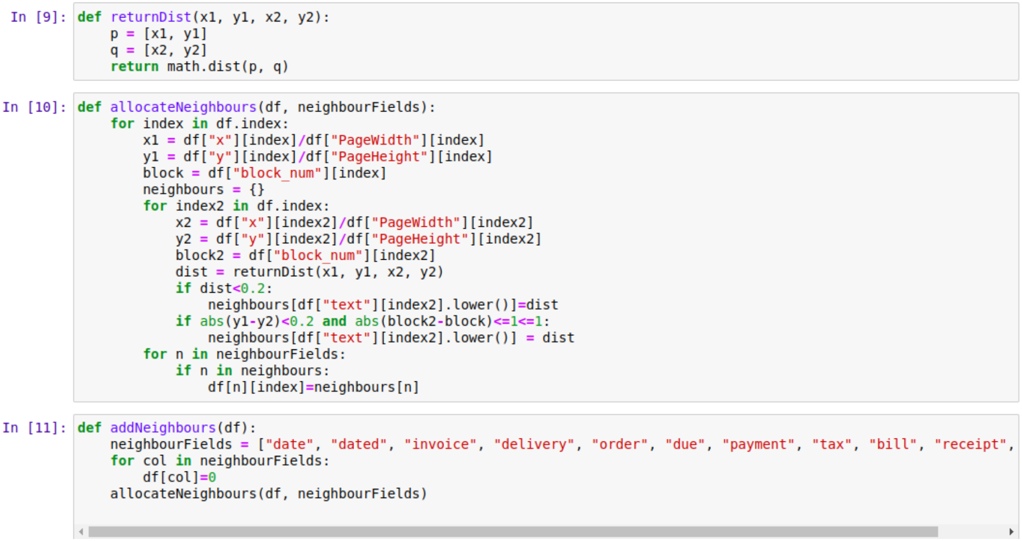
- Extracted fields having dates from this dataframe



```
In [7]: def findDate(dateDF):
            dates = []

            for index in dateDF.index:
                text = dateDF["text"][index]
                if verifyDate(text):
                    dates.append(text)
                    Dates = list(datefinder.find_dates(text))
                    dateToday = Dates[0].today()
                    if Dates[0]<=dateToday:
                        dateDF["isPast"][index]=1
                    else:
                        dateDF["isFuture"][index]=1


            return dates
```

```
In [8]: def extractDateDataFrame(df):
            dateDF = df.copy()
            for index in dateDF.index:
                text = dateDF["text"][index]
                dateDF["text"][index] = text.lower()
            dates = findDate(dateDF)
            return dateDF, dates
```

- Allocated neighbour columns with thier distance from respective date field



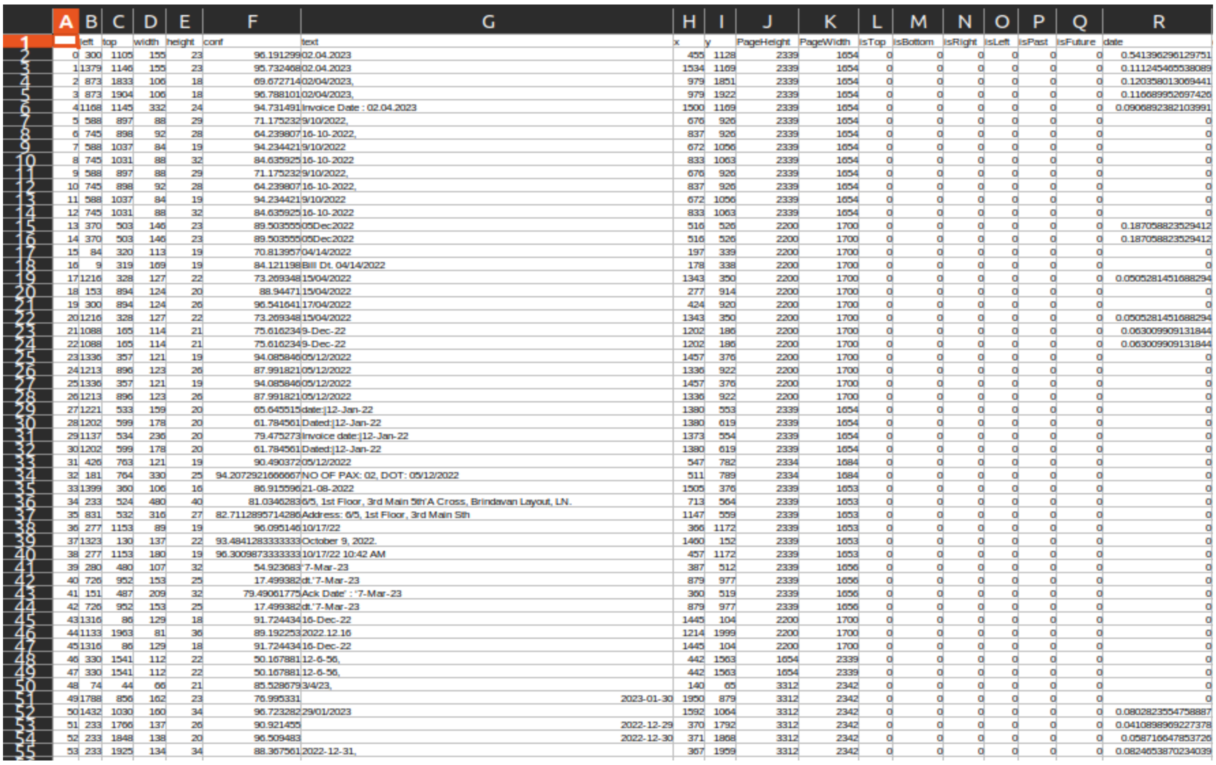
```
In [9]: def returnDist(x1, y1, x2, y2):
            p = [x1, y1]
            q = [x2, y2]
            return math.dist(p, q)
```

```
In [10]: def allocateNeighbours(df, neighbourFields):
             for index in df.index:
                 x1 = df["x"][index]/df["PageWidth"][index]
                 y1 = df["y"][index]/df["PageHeight"][index]
                 block = df["block_num"][index]
                 neighbours = {}
                 for index2 in df.index:
                     x2 = df["x"][index2]/df["PageWidth"][index2]
                     y2 = df["y"][index2]/df["PageHeight"][index2]
                     block2 = df["block_num"][index2]
                     dist = returnDist(x1, y1, x2, y2)
                     if dist<0.2:
                         neighbours[df["text"][index2].lower()]=dist
                     if abs(y1-y2)<0.2 and abs(block2-block)<=1<=1:
                         neighbours[df["text"][index2].lower()] = dist
                 for n in neighbourFields:
                     if n in neighbours:
                         df[n][index]=neighbours[n]
```

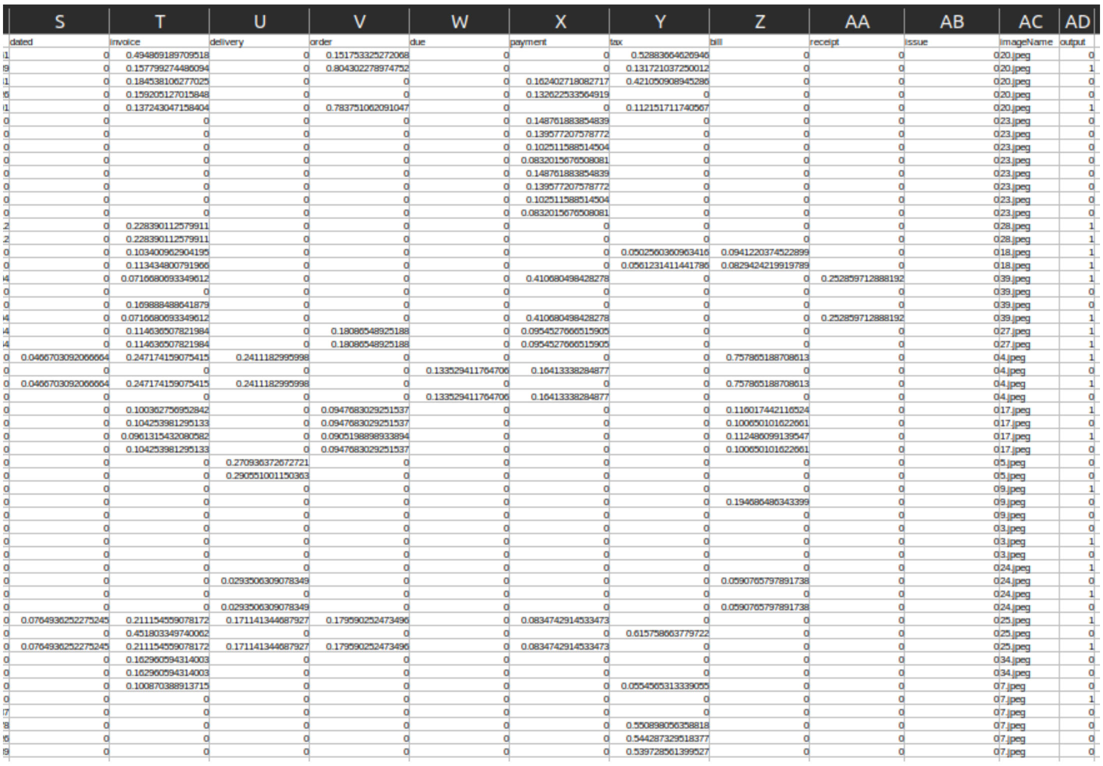
```
In [11]: def addNeighbours(df):
             neighbourFields = ["date", "dated", "invoice", "delivery", "order", "due", "payment", "tax", "bill", "receipt",
             for col in neighbourFields:
                 df[col]=0
             allocateNeighbours(df, neighbourFields)
```

- Saved this dataframe as .csv file
- Manually annotated the true(1) and false(0) invoice dates in this csv.

DATASET :



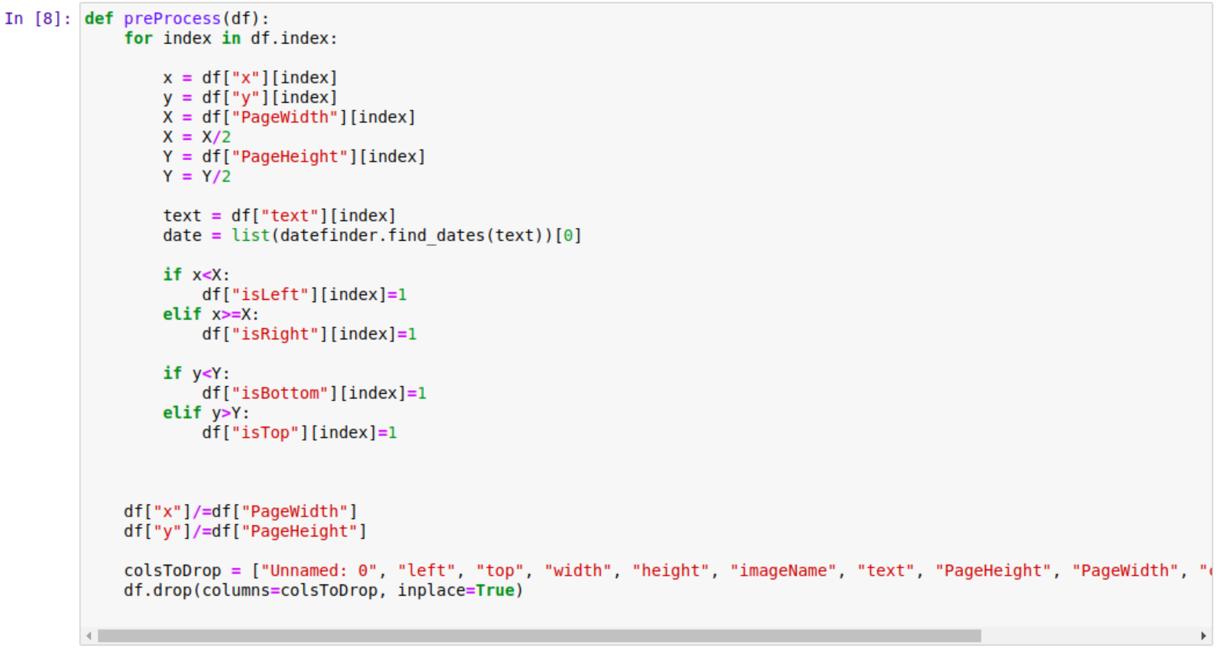
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1		left	top	width	height	conf	text	x	y	PageHeight	PageWidth	IsTop	IsBottom	IsRight	IsLeft	IsPast	IsFuture	date
2	0	300	1105	155	23	96.191299	02.04.2023	455	1128	2339	1654	0	0	0	0	0	0	0.541396296129751
3	1	1379	1146	155	23	95.732468	02.04.2023	1534	1169	2339	1654	0	0	0	0	0	0	0.111245465538089
4	2	873	1833	106	18	69.672714	02/04/2023,	979	1851	2339	1654	0	0	0	0	0	0	0.120358013069441
5	3	873	1904	106	18	96.788101	02/04/2023,	979	1922	2339	1654	0	0	0	0	0	0	0.116689952697426
6	4	1168	1145	332	24	94.731491	Invoice Date : 02.04.2023	1500	1169	2339	1654	0	0	0	0	0	0	0.0906892382103991
7	5	588	897	88	29	71.175232	9/10/2022,	676	926	2339	1654	0	0	0	0	0	0	0
8	6	745	898	92	28	64.239807	16-10-2022,	837	926	2339	1654	0	0	0	0	0	0	0
9	7	588	1037	84	19	94.234421	9/10/2022	672	1056	2339	1654	0	0	0	0	0	0	0
10	8	745	1031	88	32	84.635925	16-10-2022	833	1063	2339	1654	0	0	0	0	0	0	0
11	9	588	897	88	29	71.175232	9/10/2022,	676	926	2339	1654	0	0	0	0	0	0	0
12	10	745	898	92	28	64.239807	16-10-2022,	837	926	2339	1654	0	0	0	0	0	0	0
13	11	588	1037	84	19	94.234421	9/10/2022	672	1056	2339	1654	0	0	0	0	0	0	0
14	12	745	1031	88	32	84.635925	16-10-2022	833	1063	2339	1654	0	0	0	0	0	0	0
15	13	370	503	146	23	89.503555	05Dec2022	516	526	2200	1700	0	0	0	0	0	0	0.187058823529412
16	14	370	503	146	23	89.503555	05Dec2022	516	526	2200	1700	0	0	0	0	0	0	0.187058823529412
17	15	84	320	113	19	70.813957	04/14/2022	197	339	2200	1700	0	0	0	0	0	0	0
18	16	9	319	169	19	84.121198	Bill Dt. 04/14/2022	178	338	2200	1700	0	0	0	0	0	0	0
19	17	1216	328	127	22	73.269348	15/04/2022	1343	350	2200	1700	0	0	0	0	0	0	0.0505281451688294
20	18	153	894	124	20	88.94471	15/04/2022	277	914	2200	1700	0	0	0	0	0	0	0
21	19	300	894	124	26	96.541641	17/04/2022	424	920	2200	1700	0	0	0	0	0	0	0
22	20	1216	328	127	22	73.269348	15/04/2022	1343	350	2200	1700	0	0	0	0	0	0	0.0505281451688294
23	21	1088	165	114	21	75.616234	9-Dec-22	1202	186	2200	1700	0	0	0	0	0	0	0.063009909131844
24	22	1088	165	114	21	75.616234	9-Dec-22	1202	186	2200	1700	0	0	0	0	0	0	0.063009909131844
25	23	1336	357	121	19	94.085846	05/12/2022	1457	376	2200	1700	0	0	0	0	0	0	0
26	24	1213	896	123	26	87.991821	05/12/2022	1336	922	2200	1700	0	0	0	0	0	0	0
27	25	1336	357	121	19	94.085846	05/12/2022	1457	376	2200	1700	0	0	0	0	0	0	0
28	26	1213	896	123	26	87.991821	05/12/2022	1336	922	2200	1700	0	0	0	0	0	0	0
29	27	1221	533	159	20	65.645515	date:|12-Jan-22	1380	553	2339	1654	0	0	0	0	0	0	0
30	28	1202	599	178	20	61.784561	Dated:|12-Jan-22	1380	619	2339	1654	0	0	0	0	0	0	0
31	29	1137	534	236	20	79.475273	Invoice date:|12-Jan-22	1373	554	2339	1654	0	0	0	0	0	0	0
32	30	1202	599	178	20	61.784561	Dated:|12-Jan-22	1380	619	2339	1654	0	0	0	0	0	0	0
33	31	426	763	121	19	90.490372	05/12/2022	547	782	2334	1684	0	0	0	0	0	0	0
34	32	181	764	330	25	94.2072921666667	NO OF PAX: 02, DOT: 05/12/2022	511	789	2334	1684	0	0	0	0	0	0	0
35	33	1399	360	106	16	86.915596	21-08-2022	1505	376	2339	1653	0	0	0	0	0	0	0
36	34	233	524	480	40	81.0346283	6/5, 1st Floor, 3rd Main 5th'A Cross, Brindavan Layout, LN.	713	564	2339	1653	0	0	0	0	0	0	0
37	35	831	532	316	27	82.7112895714286	Address: 6/5, 1st Floor, 3rd Main Sth	1147	559	2339	1653	0	0	0	0	0	0	0
38	36	277	1153	89	19	96.095146	10/17/22	366	1172	2339	1653	0	0	0	0	0	0	0
39	37	1323	130	137	22	93.4841283333333	October 9, 2022.	1460	152	2339	1653	0	0	0	0	0	0	0
40	38	277	1153	180	19	96.3009873333333	10/17/22 10:42 AM	457	1172	2339	1653	0	0	0	0	0	0	0
41	39	280	480	107	32	54.923683	'7-Mar-23	387	512	2339	1656	0	0	0	0	0	0	0
42	40	726	952	153	25	17.499382	dt.'7-Mar-23	879	977	2339	1656	0	0	0	0	0	0	0
43	41	151	487	209	32	79.49061775	Ack Date' : '7-Mar-23	360	519	2339	1656	0	0	0	0	0	0	0
44	42	726	952	153	25	17.499382	dt.'7-Mar-23	879	977	2339	1656	0	0	0	0	0	0	0
45	43	1316	86	129	18	91.724434	16-Dec-22	1445	104	2200	1700	0	0	0	0	0	0	0
46	44	1133	1963	81	36	89.192253	2022.12.16	1214	1999	2200	1700	0	0	0	0	0	0	0
47	45	1316	86	129	18	91.724434	16-Dec-22	1445	104	2200	1700	0	0	0	0	0	0	0
48	46	330	1541	112	22	50.167881	12-6-56,	442	1563	1654	2339	0	0	0	0	0	0	0
49	47	330	1541	112	22	50.167881	12-6-56,	442	1563	1654	2339	0	0	0	0	0	0	0
50	48	74	44	66	21	85.528679	3/4/23,	140	65	3312	2342	0	0	0	0	0	0	0
51	49	1788	856	162	23	76.995331	2023-01-30	1950	879	3312	2342	0	0	0	0	0	0	0
52	50	1432	1030	160	34	96.723282	29/01/2023	1592	1064	3312	2342	0	0	0	0	0	0	0.0802823554758887
53	51	233	1766	137	26	90.921455	2022-12-29	370	1792	3312	2342	0	0	0	0	0	0	0.0410898969227378
54	52	233	1848	138	20	96.509483	2022-12-30	371	1868	3312	2342	0	0	0	0	0	0	0.058716647853726
55	53	233	1925	134	34	88.367561	2022-12-31,	367	1959	3312	2342	0	0	0	0	0	0	0.0824653870234039



S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
dated	invoice	delivery	order	due	payment	tax	bill	receipt	issue	imageName	output
0	0.494869189709518	0	0.151753325272068	0	0	0.52883664626946	0	0	0	20.jpeg	0
0	0.157799274486094	0	0.804302278974752	0	0	0.131721037250012	0	0	0	20.jpeg	1
0	0.184538106277025	0	0	0	0.162402718082717	0.421050908945286	0	0	0	20.jpeg	0
0	0.159205127015848	0	0	0	0.132622533564919	0	0	0	0	20.jpeg	0
0	0.137243047158404	0	0.783751062091047	0	0	0.112151711740567	0	0	0	20.jpeg	1
0	0	0	0	0	0.148761883854839	0	0	0	0	23.jpeg	0
0	0	0	0	0	0.139577207578772	0	0	0	0	23.jpeg	0
0	0	0	0	0	0.102511588514504	0	0	0	0	23.jpeg	0
0	0	0	0	0	0.0832015676508081	0	0	0	0	23.jpeg	0
0	0	0	0	0	0.148761883854839	0	0	0	0	23.jpeg	0
0	0	0	0	0	0.139577207578772	0	0	0	0	23.jpeg	0
0	0	0	0	0	0.102511588514504	0	0	0	0	23.jpeg	0
0	0	0	0	0	0.0832015676508081	0	0	0	0	23.jpeg	0
0	0.228390112579911	0	0	0	0	0	0	0	0	28.jpeg	1
0	0.228390112579911	0	0	0	0	0	0	0	0	28.jpeg	1
0	0.103400962904195	0	0	0	0	0.0502560360963416	0.0941220374522899	0	0	18.jpeg	1
0	0.113434800791966	0	0	0	0	0.0561231411441786	0.0829424219919789	0	0	18.jpeg	1
0	0.0716680693349612	0	0	0	0.410680498428278	0	0	0.252859712888192	0	39.jpeg	1
0	0	0	0	0	0	0	0	0	0	39.jpeg	0
0	0.169888488641879	0	0	0	0	0	0	0	0	39.jpeg	0
0	0.0716680693349612	0	0	0	0.410680498428278	0	0	0.252859712888192	0	39.jpeg	1
0	0.114636507821984	0	0.18086548925188	0	0.0954527666515905	0	0	0	0	27.jpeg	1
0	0.114636507821984	0	0.18086548925188	0	0.0954527666515905	0	0	0	0	27.jpeg	1
0.0466703092066664	0.247174159075415	0.2411182995998	0	0	0	0	0.757865188708613	0	0	4.jpeg	1
0	0	0	0	0.133529411764706	0.16413338284877	0	0	0	0	4.jpeg	0
0.0466703092066664	0.247174159075415	0.2411182995998	0	0	0	0	0.757865188708613	0	0	4.jpeg	1
0	0	0	0	0.133529411764706	0.16413338284877	0	0	0	0	4.jpeg	0
0	0.100362756952842	0	0.0947683029251537	0	0	0	0.116017442116524	0	0	17.jpeg	1
0	0.104253981295133	0	0.0947683029251537	0	0	0	0.100650101622661	0	0	17.jpeg	0
0	0.0961315432080582	0	0.0905198898933894	0	0	0	0.112486099139547	0	0	17.jpeg	1
0	0.104253981295133	0	0.0947683029251537	0	0	0	0.100650101622661	0	0	17.jpeg	0
0	0	0.270936372672721	0	0	0	0	0	0	0	5.jpeg	0
0	0	0.290551001150363	0	0	0	0	0	0	0	5.jpeg	0
0	0	0	0	0	0	0	0	0	0	9.jpeg	1
0	0	0	0	0	0	0	0.194686486343399	0	0	9.jpeg	0
0	0	0	0	0	0	0	0	0	0	9.jpeg	0
0	0	0	0	0	0	0	0	0	0	3.jpeg	0
0	0	0	0	0	0	0	0	0	0	3.jpeg	1
0	0	0	0	0	0	0	0	0	0	3.jpeg	0
0	0	0	0	0	0	0	0	0	0	24.jpeg	1
0	0	0.0293506309078349	0	0	0	0	0.0590765797891738	0	0	24.jpeg	0
0	0	0	0	0	0	0	0	0	0	24.jpeg	1
0	0	0.0293506309078349	0	0	0	0	0.0590765797891738	0	0	24.jpeg	0
0.0764936252275245	0.211154559078172	0.171141344687927	0.179590252473496	0	0.0834742914533473	0	0	0	0	25.jpeg	1
0	0.451803349740062	0	0	0	0	0.615758663779722	0	0	0	25.jpeg	0
0.0764936252275245	0.211154559078172	0.171141344687927	0.179590252473496	0	0.0834742914533473	0	0	0	0	25.jpeg	1
0	0.162960594314003	0	0	0	0	0	0	0	0	34.jpeg	0
0	0.162960594314003	0	0	0	0	0	0	0	0	34.jpeg	0
0	0.100870388913715	0	0	0	0	0.0554565313339055	0	0	0	7.jpeg	0
0	0	0	0	0	0	0	0	0	0	7.jpeg	1
0	0	0	0	0	0	0	0	0	0	7.jpeg	0
0	0	0	0	0	0	0.550898056358818	0	0	0	7.jpeg	0
0	0	0	0	0	0	0.544287329518377	0	0	0	7.jpeg	0
0	0	0	0	0	0	0.539728561399527	0	0	0	7.jpeg	0

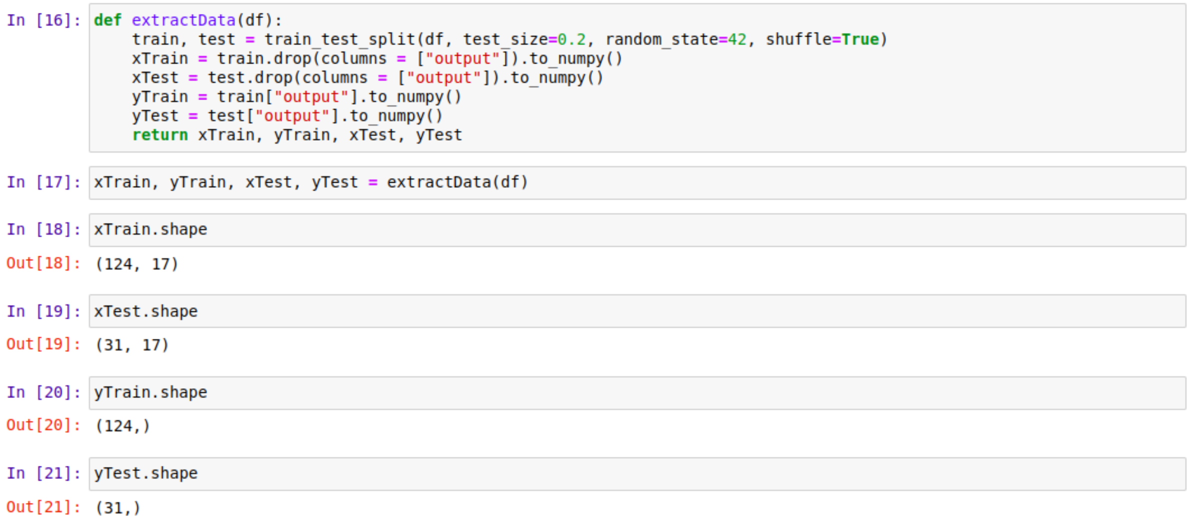
MODEL PREPERATION :

- Read this .csv dataset
- PreProcessed and normalized the dataset and added zonal info to dates



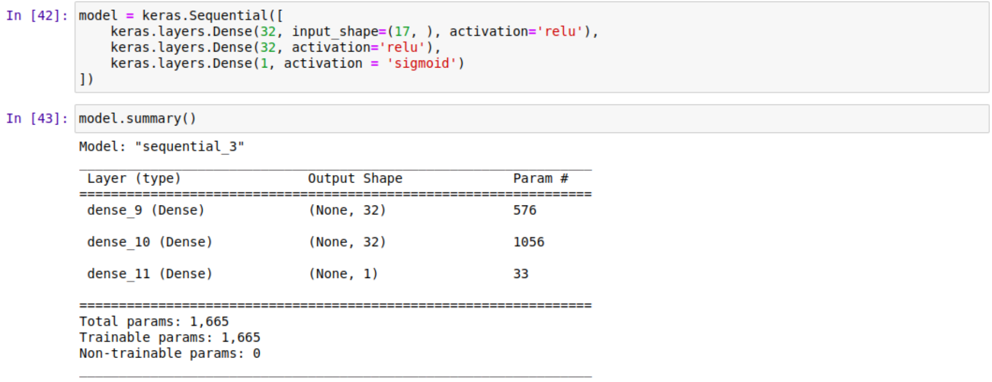
```
In [8]: def preProcess(df):
            for index in df.index:

                x = df["x"][index]
                y = df["y"][index]
                X = df["PageWidth"][index]
                X = X/2
                Y = df["PageHeight"][index]
                Y = Y/2

                text = df["text"][index]
                date = list(datefinder.find_dates(text))[0]

                if x<X:
                    df["isLeft"][index]=1
                elif x>=X:
                    df["isRight"][index]=1

                if y<Y:
                    df["isBottom"][index]=1
                elif y>Y:
                    df["isTop"][index]=1



            df["x"]/=df["PageWidth"]
            df["y"]/=df["PageHeight"]

            colsToDrop = ["Unnamed: 0", "left", "top", "width", "height", "imageName", "text", "PageHeight", "PageWidth", "
            df.drop(columns=colsToDrop, inplace=True)
```

- Divided this dataset into training and testing sets



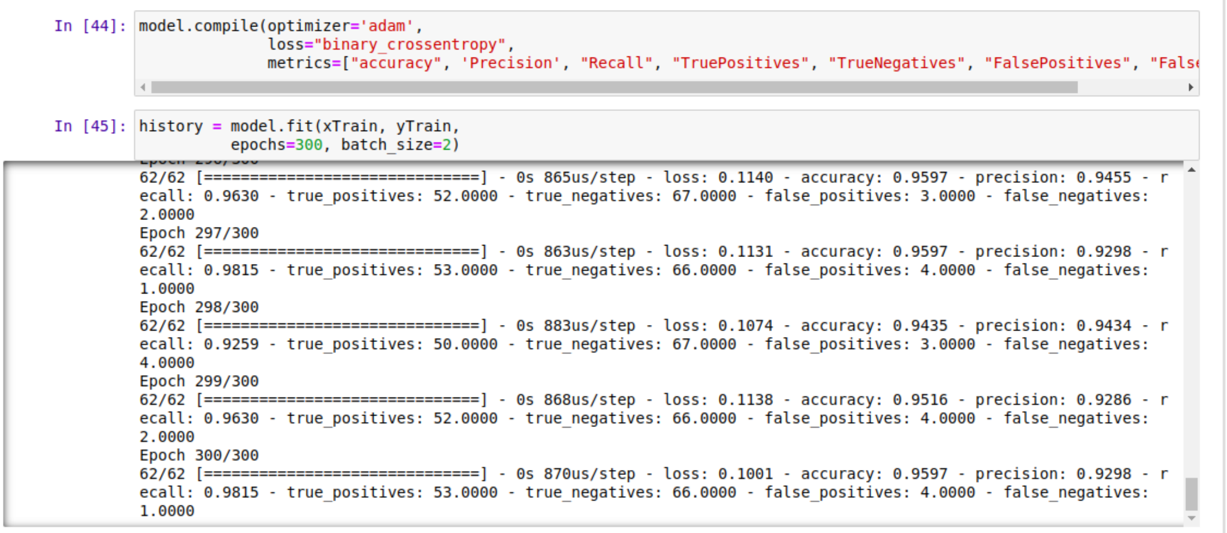
```
In [16]: def extractData(df):
             train, test = train_test_split(df, test_size=0.2, random_state=42, shuffle=True)
             xTrain = train.drop(columns = ["output"]).to_numpy()
             xTest = test.drop(columns = ["output"]).to_numpy()
             yTrain = train["output"].to_numpy()
             yTest = test["output"].to_numpy()
             return xTrain, yTrain, xTest, yTest

In [17]: xTrain, yTrain, xTest, yTest = extractData(df)

In [18]: xTrain.shape

Out[18]: (124, 17)

In [19]: xTest.shape

Out[19]: (31, 17)

In [20]: yTrain.shape

Out[20]: (124,)

In [21]: yTest.shape

Out[21]: (31,)
```

- Prepared a keras neural network to train on this dataset



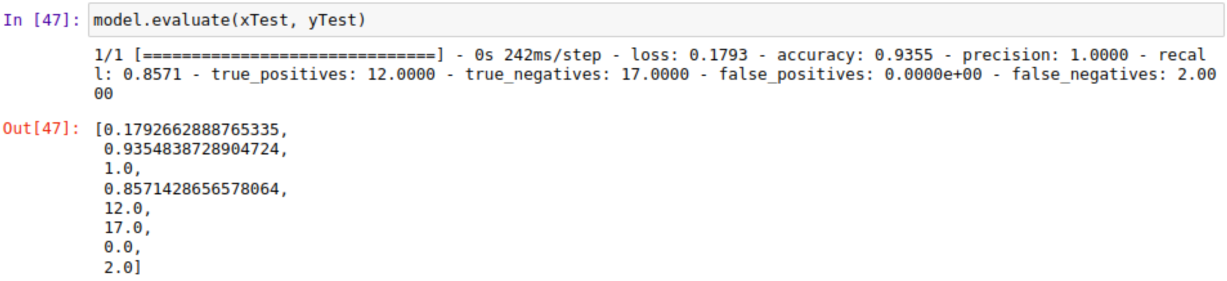
```
In [42]: model = keras.Sequential([
             keras.layers.Dense(32, input_shape=(17, ), activation='relu'),
             keras.layers.Dense(32, activation='relu'),
             keras.layers.Dense(1, activation = 'sigmoid')
         ])

In [43]: model.summary()

         Model: "sequential_3"
         _________________________________________________________________
          Layer (type)                Output Shape              Param #
         =================================================================
          dense_9 (Dense)             (None, 32)                576

          dense_10 (Dense)            (None, 32)                1056

          dense_11 (Dense)            (None, 1)                 33

         =================================================================
         Total params: 1,665
         Trainable params: 1,665
         Non-trainable params: 0
         _________________________________________________________________
```

- Used loss function “binary_crossentopy” and “adam” optimizer to compile this model and trained for 300 epochs



```
In [44]: model.compile(optimizer='adam',
                       loss="binary_crossentropy",
                       metrics=["accuracy", 'Precision', "Recall", "TruePositives", "TrueNegatives", "FalsePositives", "False

In [45]: history = model.fit(xTrain, yTrain,
                   epochs=300, batch_size=2)

         Epoch 296/300
         62/62 [==============================] - 0s 865us/step - loss: 0.1140 - accuracy: 0.9597 - precision: 0.9455 - r
         ecall: 0.9630 - true_positives: 52.0000 - true_negatives: 67.0000 - false_positives: 3.0000 - false_negatives:
         2.0000
         Epoch 297/300
         62/62 [==============================] - 0s 863us/step - loss: 0.1131 - accuracy: 0.9597 - precision: 0.9298 - r
         ecall: 0.9815 - true_positives: 53.0000 - true_negatives: 66.0000 - false_positives: 4.0000 - false_negatives:
         1.0000
         Epoch 298/300
         62/62 [==============================] - 0s 883us/step - loss: 0.1074 - accuracy: 0.9435 - precision: 0.9434 - r
         ecall: 0.9259 - true_positives: 50.0000 - true_negatives: 67.0000 - false_positives: 3.0000 - false_negatives:
         4.0000
         Epoch 299/300
         62/62 [==============================] - 0s 868us/step - loss: 0.1138 - accuracy: 0.9516 - precision: 0.9286 - r
         ecall: 0.9630 - true_positives: 52.0000 - true_negatives: 66.0000 - false_positives: 4.0000 - false_negatives:
         2.0000
         Epoch 300/300
         62/62 [==============================] - 0s 870us/step - loss: 0.1001 - accuracy: 0.9597 - precision: 0.9298 - r
         ecall: 0.9815 - true_positives: 53.0000 - true_negatives: 66.0000 - false_positives: 4.0000 - false_negatives:
         1.0000
```

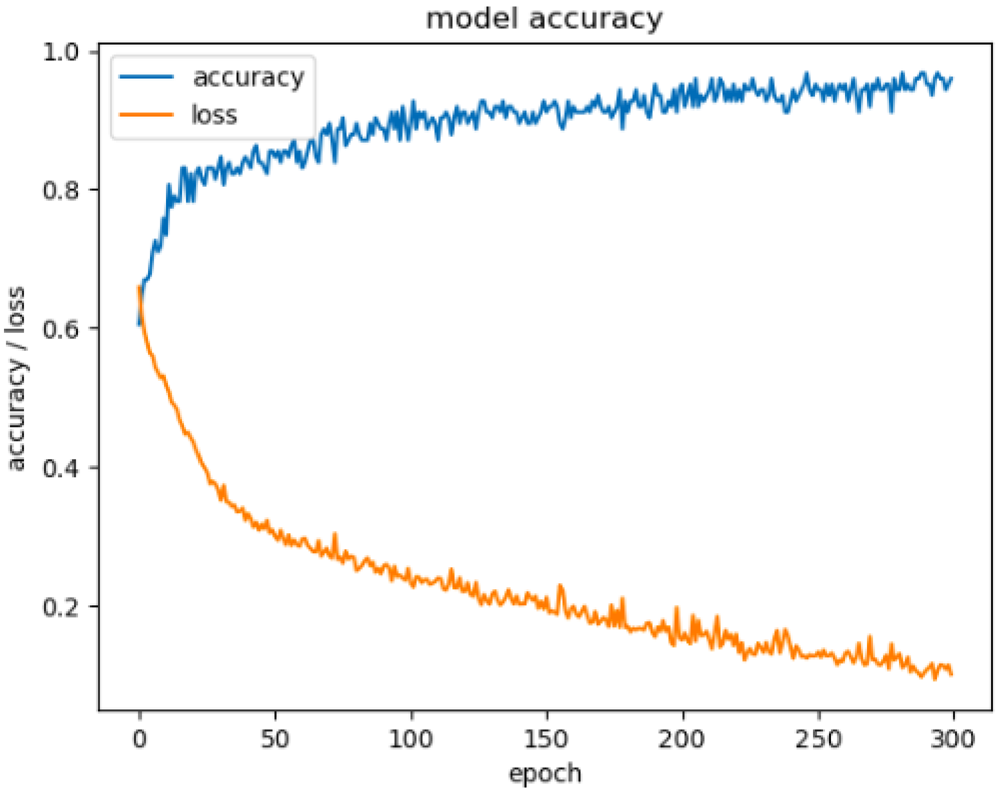
- Testing accuracy came out to be 93.55%



```
In [47]: model.evaluate(xTest, yTest)

         1/1 [==============================] - 0s 242ms/step - loss: 0.1793 - accuracy: 0.9355 - precision: 1.0000 - recal
         l: 0.8571 - true_positives: 12.0000 - true_negatives: 17.0000 - false_positives: 0.0000e+00 - false_negatives: 2.00
         00

Out[47]: [0.1792662888765335,
          0.9354838728904724,
          1.0,
          0.8571428656578064,
          12.0,
          17.0,
          0.0,
          2.0]
```

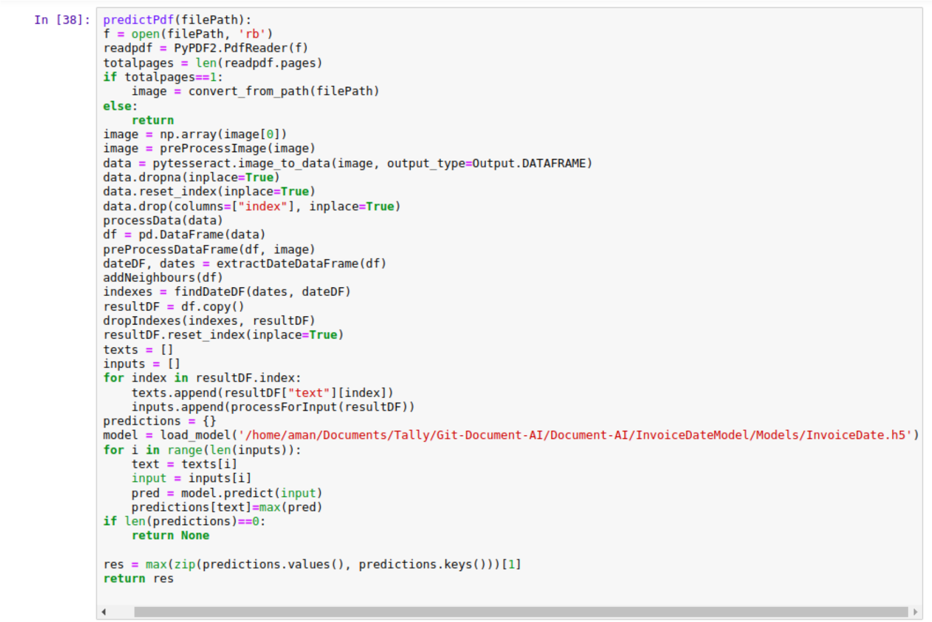
- Saved this model as .h5 for further inference

MODEL TRAINING VISUALIZATION :

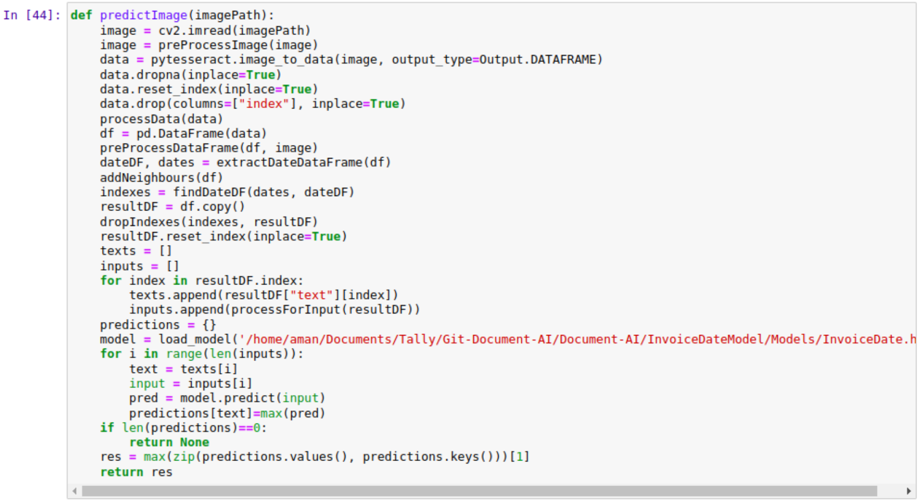


INFERENCE :

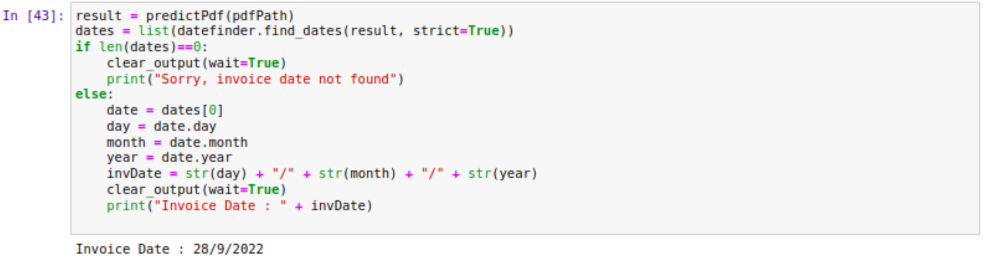
- Document was read and preprocessed to fit the trained model
- Saved model was loaded
- Model was used for predictions



```
In [38]: predictPdf(filePath):
f = open(filePath, 'rb')
readpdf = PyPDF2.PdfReader(f)
totalpages = len(readpdf.pages)
if totalpages==1:
    image = convert_from_path(filePath)
else:
    return
image = np.array(image[0])
image = preProcessImage(image)
data = pytesseract.image_to_data(image, output_type=Output.DATAFRAME)
data.dropna(inplace=True)
data.reset_index(inplace=True)
data.drop(columns=["index"], inplace=True)
processData(data)
df = pd.DataFrame(data)
preProcessDataFrame(df, image)
dateDF, dates = extractDateDataFrame(df)
addNeighbours(df)
indexes = findDateDF(dates, dateDF)
resultDF = df.copy()
dropIndexes(indexes, resultDF)
resultDF.reset_index(inplace=True)
texts = []
inputs = []
for index in resultDF.index:
    texts.append(resultDF["text"][index])
    inputs.append(processForInput(resultDF))
predictions = {}
model = load_model('/home/aman/Documents/Tally/Git-Document-AI/Document-AI/InvoiceDateModel/Models/InvoiceDate.h5')
for i in range(len(inputs)):
    text = texts[i]
    input = inputs[i]
    pred = model.predict(input)
    predictions[text]=max(pred)
if len(predictions)==0:
    return None

res = max(zip(predictions.values(), predictions.keys()))[1]
return res
```

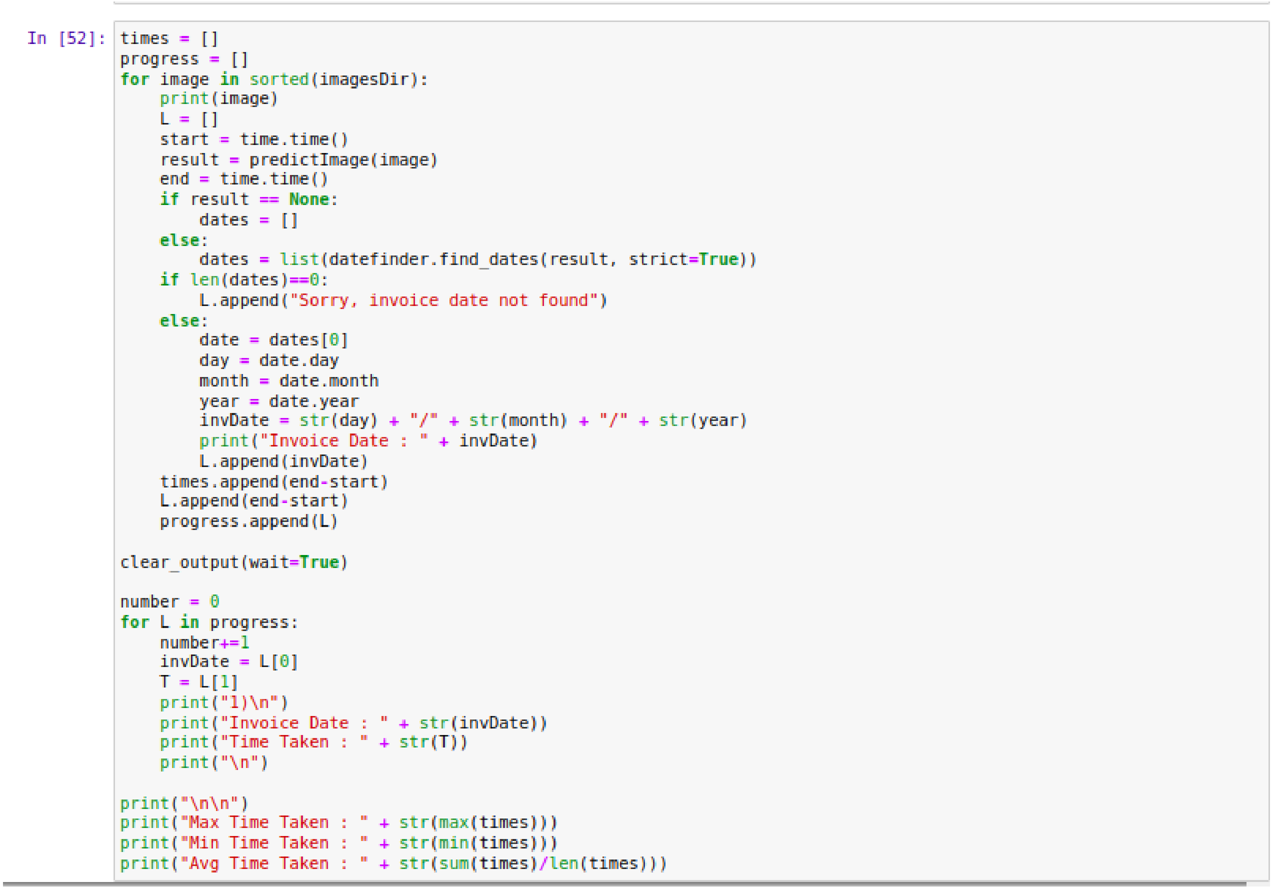


```
In [44]: def predictImage(imagePath):
    image = cv2.imread(imagePath)
    image = preProcessImage(image)
    data = pytesseract.image_to_data(image, output_type=Output.DATAFRAME)
    data.dropna(inplace=True)
    data.reset_index(inplace=True)
    data.drop(columns=["index"], inplace=True)
    processData(data)
    df = pd.DataFrame(data)
    preProcessDataFrame(df, image)
    dateDF, dates = extractDateDataFrame(df)
    addNeighbours(df)
    indexes = findDateDF(dates, dateDF)
    resultDF = df.copy()
    dropIndexes(indexes, resultDF)
    resultDF.reset_index(inplace=True)
    texts = []
    inputs = []
    for index in resultDF.index:
        texts.append(resultDF["text"][index])
        inputs.append(processForInput(resultDF))
    predictions = {}
    model = load_model('/home/aman/Documents/Tally/Git-Document-AI/Document-AI/InvoiceDateModel/Models/InvoiceDate.h
    for i in range(len(inputs)):
        text = texts[i]
        input = inputs[i]
        pred = model.predict(input)
        predictions[text]=max(pred)
    if len(predictions)==0:
        return None
    res = max(zip(predictions.values(), predictions.keys()))[1]
    return res
```



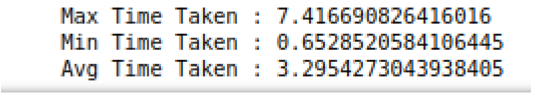
```
In [43]: result = predictPdf(pdfPath)
dates = list(datefinder.find_dates(result, strict=True))
if len(dates)==0:
    clear_output(wait=True)
    print("Sorry, invoice date not found")
else:
    date = dates[0]
    day = date.day
    month = date.month
    year = date.year
    invDate = str(day) + "/" + str(month) + "/" + str(year)
    clear_output(wait=True)
    print("Invoice Date : " + invDate)

Invoice Date : 28/9/2022
```

- Model was used on a folder on 50 images to test its performance.



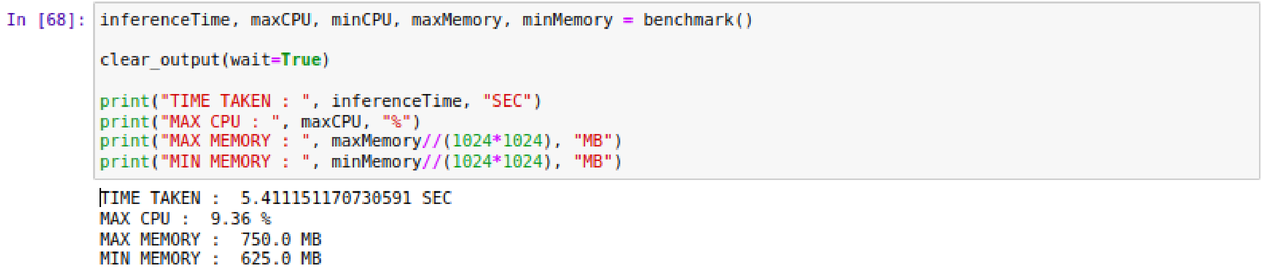
```
In [52]: times = []
         progress = []
         for image in sorted(imagesDir):
             print(image)
             L = []
             start = time.time()
             result = predictImage(image)
             end = time.time()
             if result == None:
                 dates = []
             else:
                 dates = list(datefinder.find_dates(result, strict=True))
             if len(dates)==0:
                 L.append("Sorry, invoice date not found")
             else:
                 date = dates[0]
                 day = date.day
                 month = date.month
                 year = date.year
                 invDate = str(day) + "/" + str(month) + "/" + str(year)
                 print("Invoice Date : " + invDate)
                 L.append(invDate)
             times.append(end-start)
             L.append(end-start)
             progress.append(L)

         clear_output(wait=True)

         number = 0
         for L in progress:
             number+=1
             invDate = L[0]
             T = L[1]
             print("1)\n")
             print("Invoice Date : " + str(invDate))
             print("Time Taken : " + str(T))
             print("\n")

         print("\n\n")
         print("Max Time Taken : " + str(max(times)))
         print("Min Time Taken : " + str(min(times)))
         print("Avg Time Taken : " + str(sum(times)/len(times)))
```

- Output was as follows



```
Max Time Taken : 7.416690826416016
Min Time Taken : 0.6528520584106445
Avg Time Taken : 3.2954273043938405
```

- A benchmark was prepared for monitoring CPU and RAM usage.



```
In [68]: inferenceTime, maxCPU, minCPU, maxMemory, minMemory = benchmark()

         clear_output(wait=True)

         print("TIME TAKEN : ", inferenceTime, "SEC")
         print("MAX CPU : ", maxCPU, "%")
         print("MAX MEMORY : ", maxMemory//(1024*1024), "MB")
         print("MIN MEMORY : ", minMemory//(1024*1024), "MB")

         TIME TAKEN :  5.411151170730591 SEC
         MAX CPU :  9.36 %
         MAX MEMORY :  750.0 MB
         MIN MEMORY :  625.0 MB
```