In [1]:

```
import os
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
train = pd.read_csv('C:Documents/titanic_train.csv')
```

In [3]:

```
train.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Er
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [4]:

```
train.info()
```

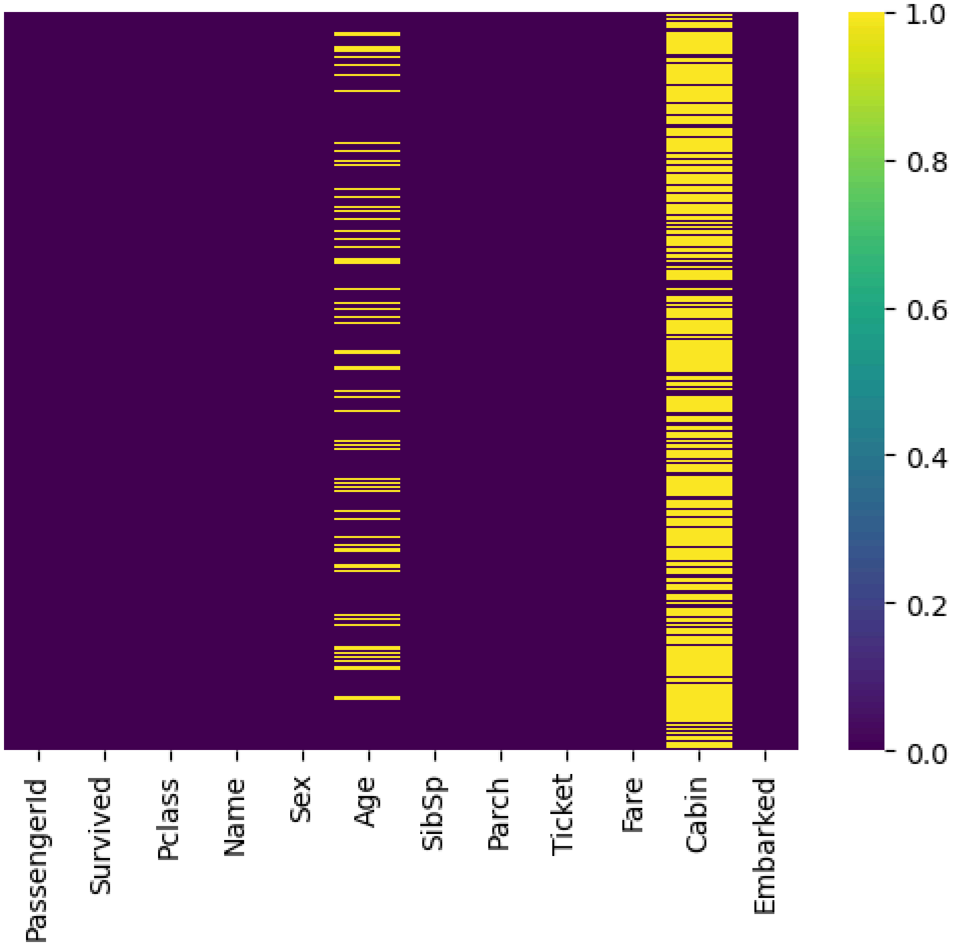
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [5]:

```
sns.heatmap(train.isnull(), yticklabels=False, cmap = 'viridis' )
```

Out[5]:

```
<Axes: >
```

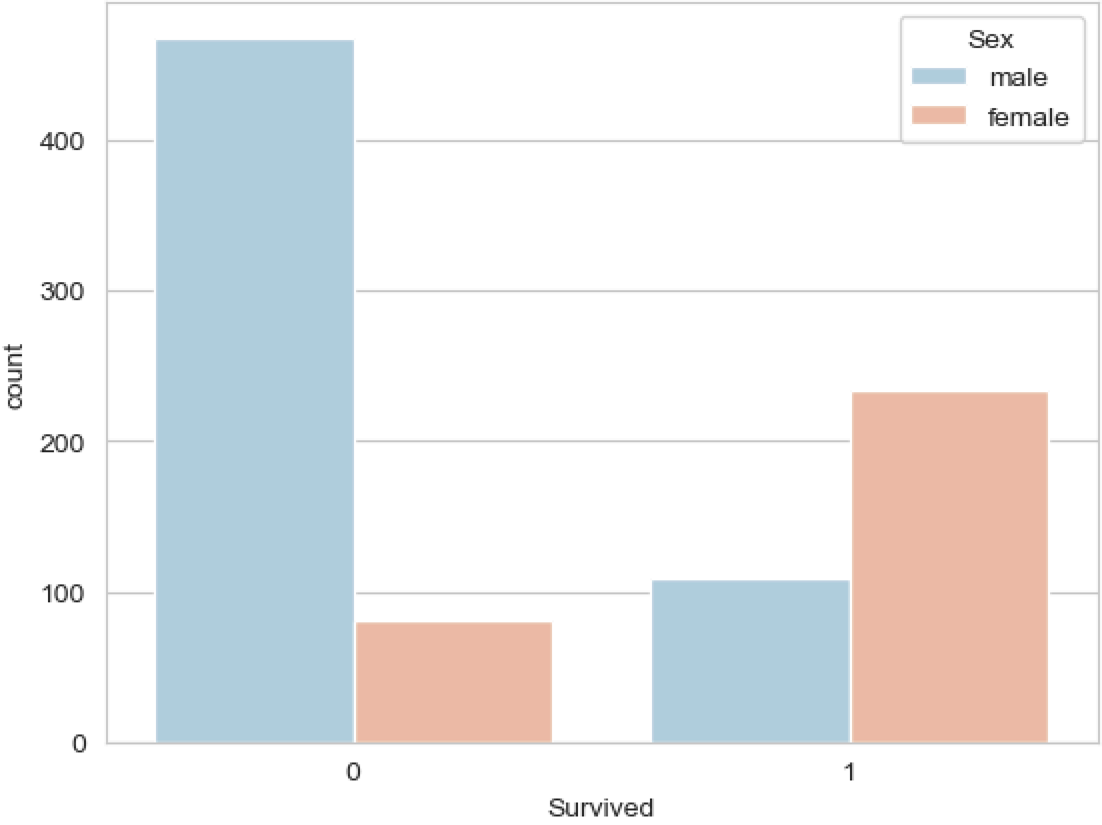


In [6]:

```
sns.set_style('whitegrid')
sns.countplot(x= 'Survived', hue = 'Sex', data=train, palette= 'RdBu_r')
```

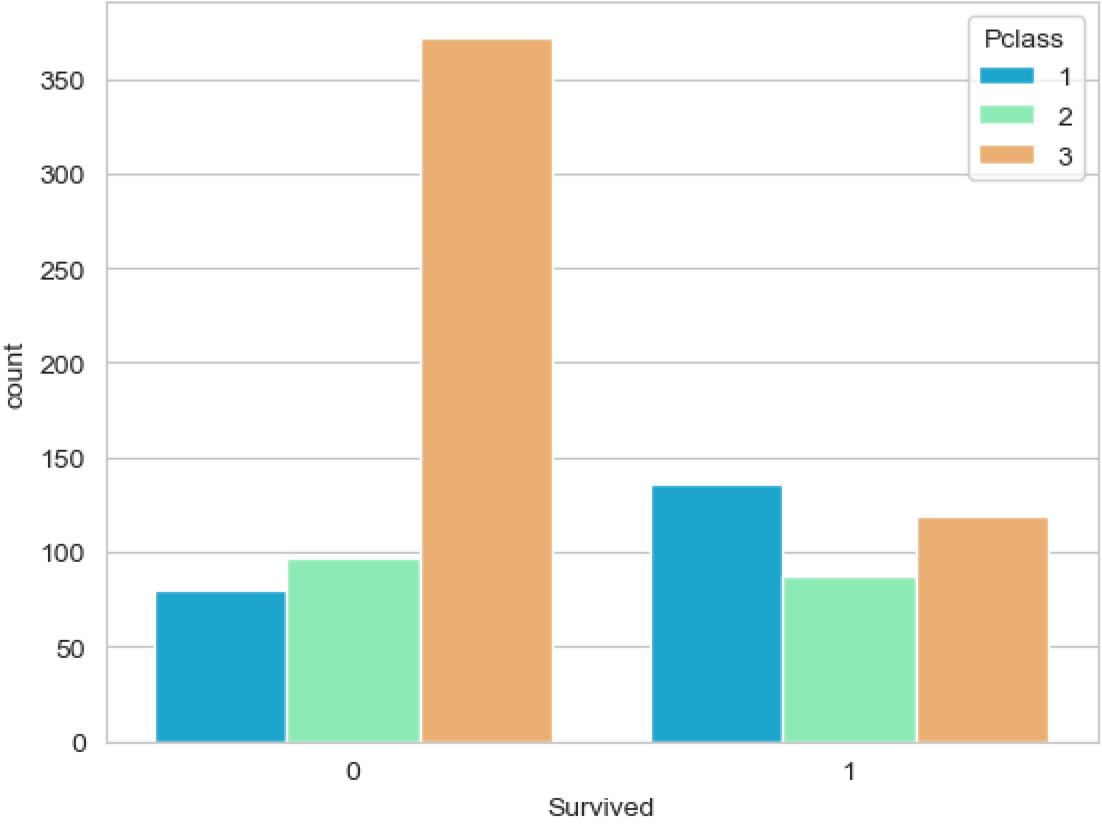
Out[6]:

```
<Axes: xlabel='Survived', ylabel='count'>
```



In [7]:
```
sns.set_style('whitegrid')
sns.countplot(x='Survived', hue='Pclass', data=train,palette='rainbow')
```

Out[7]:
```
<Axes: xlabel='Survived', ylabel='count'>
```

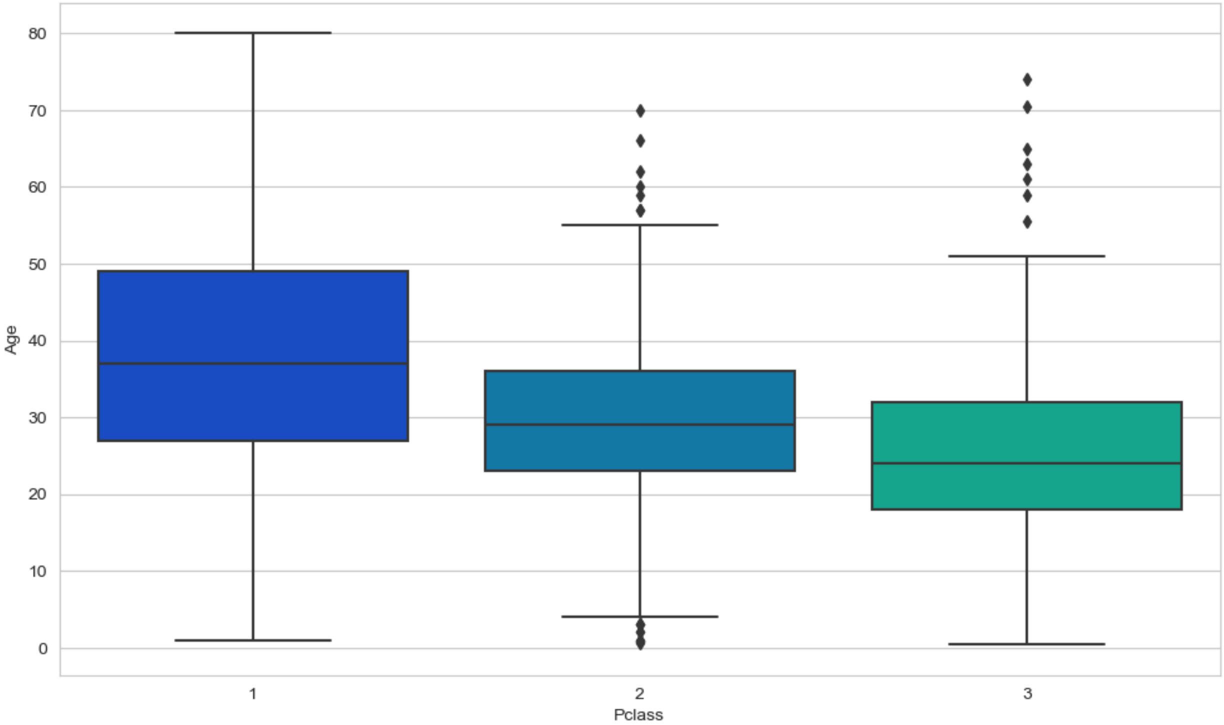


In [8]:

```
plt.figure(figsize=(12,7))
sns.boxplot(x= 'Pclass', y='Age',data=train, palette = 'winter')
```

Out[8]:

```
<Axes: xlabel='Pclass', ylabel='Age'>
```



In [9]:

```
def impute_age(cols):
    Age = cols[0]
    Pclass=cols[1]

    if pd.isnull(Age):
        if Pclass ==1:
            return 37
        elif Pclass == 2:
            return 29
        else:
            return 24
    else:
        return Age
```

In [10]:

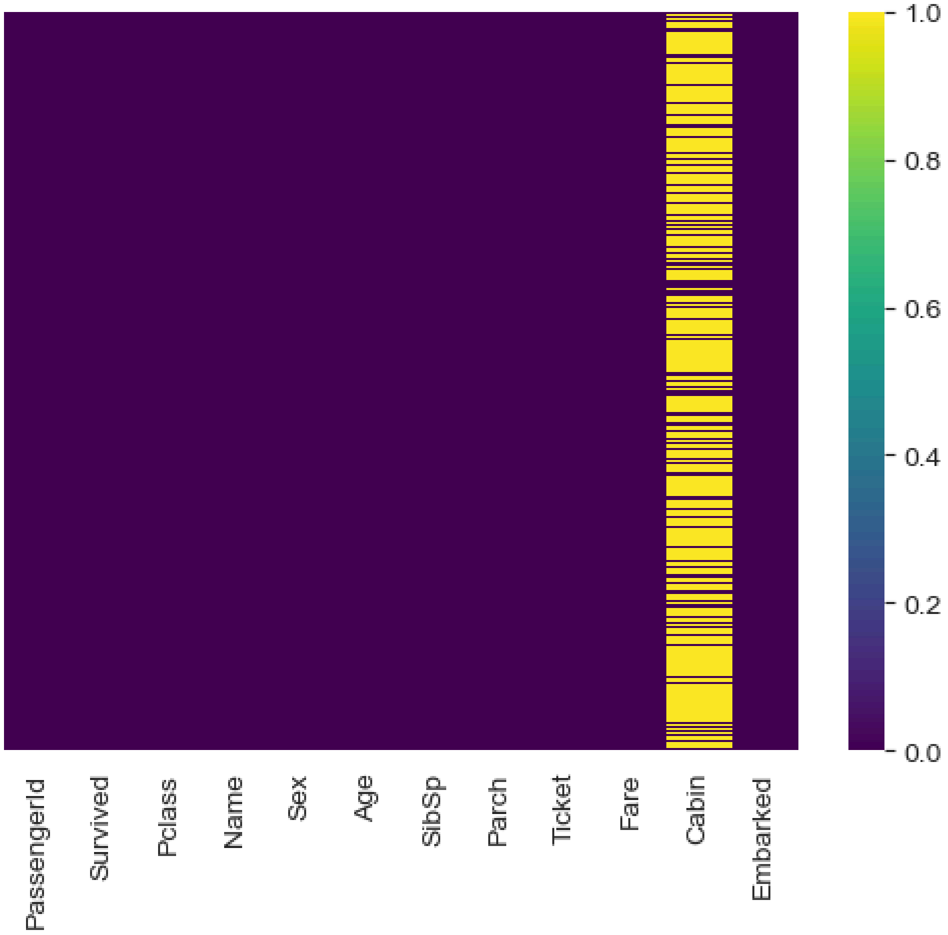
```
train ['Age'] = train [['Age', 'Pclass']].apply(impute_age, axis=1)
```

In [11]:

```
sns.heatmap(train.isnull(), yticklabels=False, cmap='viridis')
```

Out[11]:

```
<Axes: >
```



In [12]:

```
train.drop('Cabin', axis = 1, inplace = True)
train.dropna(inplace=True)
train.head()
```

Out[12]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarkec
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S

In [13]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 889 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  889 non-null    int64  
 1   Survived     889 non-null    int64  
 2   Pclass       889 non-null    int64  
 3   Name         889 non-null    object 
 4   Sex          889 non-null    object 
 5   Age          889 non-null    float64
 6   SibSp        889 non-null    int64  
 7   Parch        889 non-null    int64  
 8   Ticket       889 non-null    object 
 9   Fare         889 non-null    float64
 10  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

In [14]:
```
sex= pd.get_dummies(train['Sex'],dtype=int, drop_first=True)
embark = pd.get_dummies(train['Embarked'],dtype=int, drop_first=True)
```

In [15]:
```
train.drop(['Sex', 'Embarked', 'Name', 'Ticket'], axis = 1, inplace=True)
```

In [16]:
```
train = pd.concat([train,sex, embark], axis = 1)
```

In [17]:
```
train.head(10)
```

Out[17]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	1	0	1
1	2	1	1	38.0	1	0	71.2833	0	0	0
2	3	1	3	26.0	0	0	7.9250	0	0	1
3	4	1	1	35.0	1	0	53.1000	0	0	1
4	5	0	3	35.0	0	0	8.0500	1	0	1
5	6	0	3	24.0	0	0	8.4583	1	1	0
6	7	0	1	54.0	0	0	51.8625	1	0	1
7	8	0	3	2.0	3	1	21.0750	1	0	1
8	9	1	3	27.0	0	2	11.1333	0	0	1
9	10	1	2	14.0	1	0	30.0708	0	0	0

In [18]:

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'],
                                                    test_size = 0.3,
                                                    random_state=101)
```

In [19]:

```python
#Training and Predicting

from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression()

logmodel.fit (X_train, y_train)
```

```
C:\Users\USER\Downloads\IGAD\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[19]:

```
▾ LogisticRegression
LogisticRegression()
```

In [20]:

```python
predictions =  logmodel.predict(X_test)
```

In [21]:

```python
#Evaluation

from sklearn import metrics

print ("Accuracy: ", metrics.accuracy_score(y_test,predictions))
print ("Precision: ", metrics.precision_score(y_test,predictions))
print ("Recall: ", metrics.recall_score(y_test,predictions))
```

```
Accuracy:  0.797752808988764
Precision:  0.8125
Recall:  0.625
```

In [22]:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, predictions)
print (cm)
```

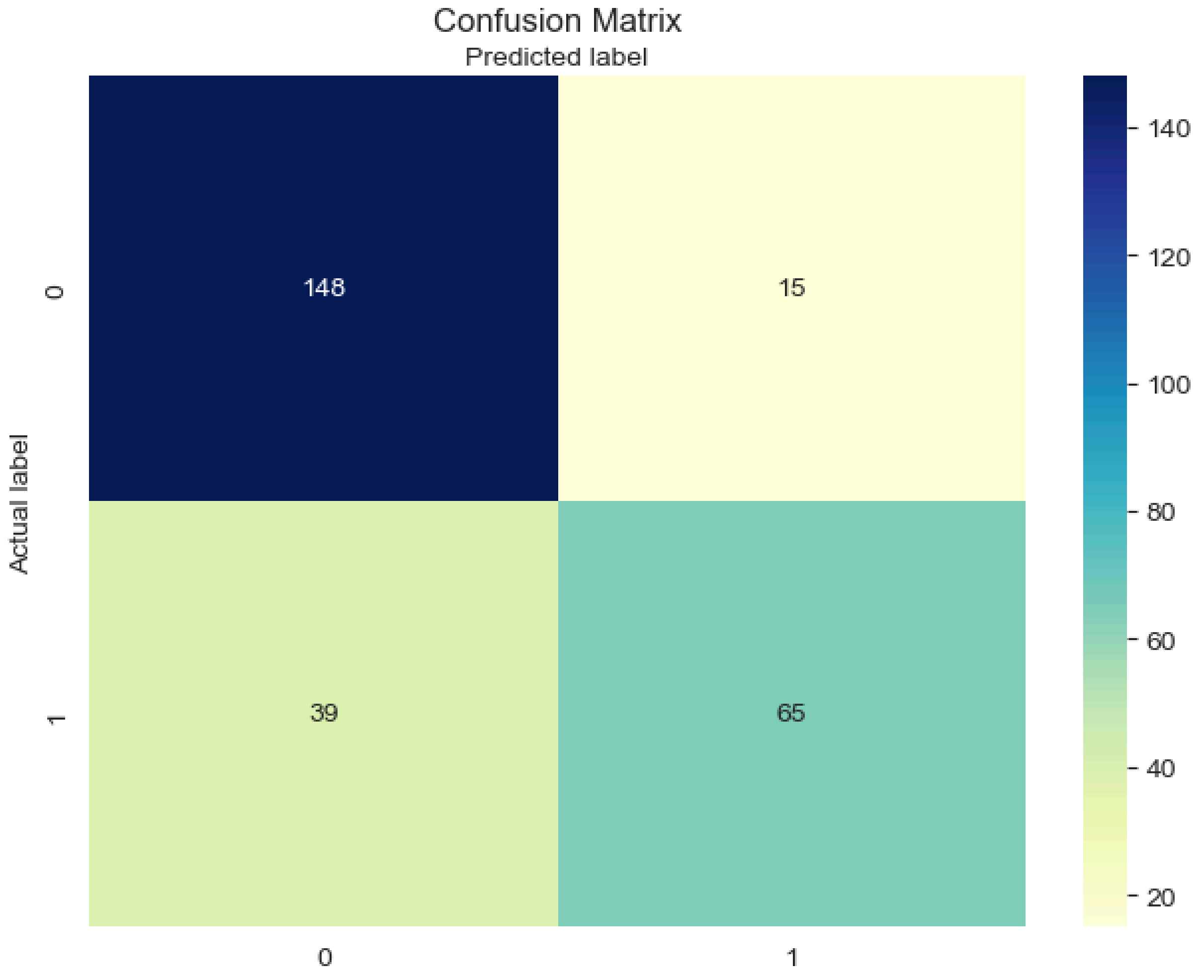
```
[[148  15]
 [ 39  65]]
```

In [23]:

```
class_names = [0,1]

fig,ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks,class_names)
plt.yticks(tick_marks,class_names)

sns.heatmap(pd.DataFrame(cm), annot=True, cmap ='YlGnBu', fmt = 'g')
ax.xaxis.set_label_position("top")

plt.tight_layout()
plt.title("Confusion Matrix")
plt.ylabel("Actual label")
plt.xlabel("Predicted label")
```

Out[23]:

```
Text(0.5, 427.9555555555555, 'Predicted\xa0label')
```



MNIST DATA SET

In [24]:

```
from sklearn.datasets import fetch_openml
X,y = fetch_openml('mnist_784', version = 1, return_X_y=True)
```

```
C:\Users\USER\Downloads\IGAD\Lib\site-packages\sklearn\datasets\_openml.py:1002: Futu
reWarning: The default value of `parser` will change from `'liac-arff'` to `'auto'` i
n 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `ImportErro
r` will be raised from 1.4 if the dataset is dense and pandas is not installed. Note
that the pandas parser may return different data types. See the Notes Section in fetc
h_openml's API doc for details.
  warn(
```

In [25]:

```
X.shape
```

Out[25]:

```
(70000, 784)
```

In [26]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test= train_test_split(X,
                                                  y,
                                                  test_size = 1/7,
                                                  random_state=0)
```

In [27]:

```
X_train.shape
```

Out[27]:

```
(60000, 784)
```

In [28]:

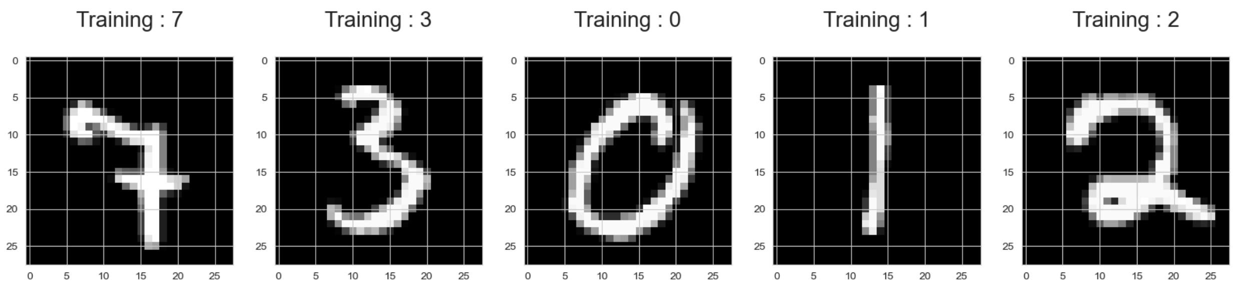
```
y_test.shape
```

Out[28]:

```
(10000,)
```

In [29]:

```
plt.figure(figsize=(20,4))
for index in range (5):
    plt.subplot(1,5, index+1)
    plt.imshow(X_train.to_numpy()[index].reshape((28,28)), cmap=plt.cm.gray)
    plt.title('Training : %i\n' %int(y_train.to_numpy()[index]), fontsize=20)
```



In [30]:

```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression()

logmodel.fit (X_train, y_train)
```

```
C:\Users\USER\Downloads\IGAD\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[30]:

```
▾ LogisticRegression

LogisticRegression()
```

In [31]:

```
predictions = logmodel.predict(X_test)
```

In [32]:

```
score = logmodel.score(X_test,y_test)
print (score)
```

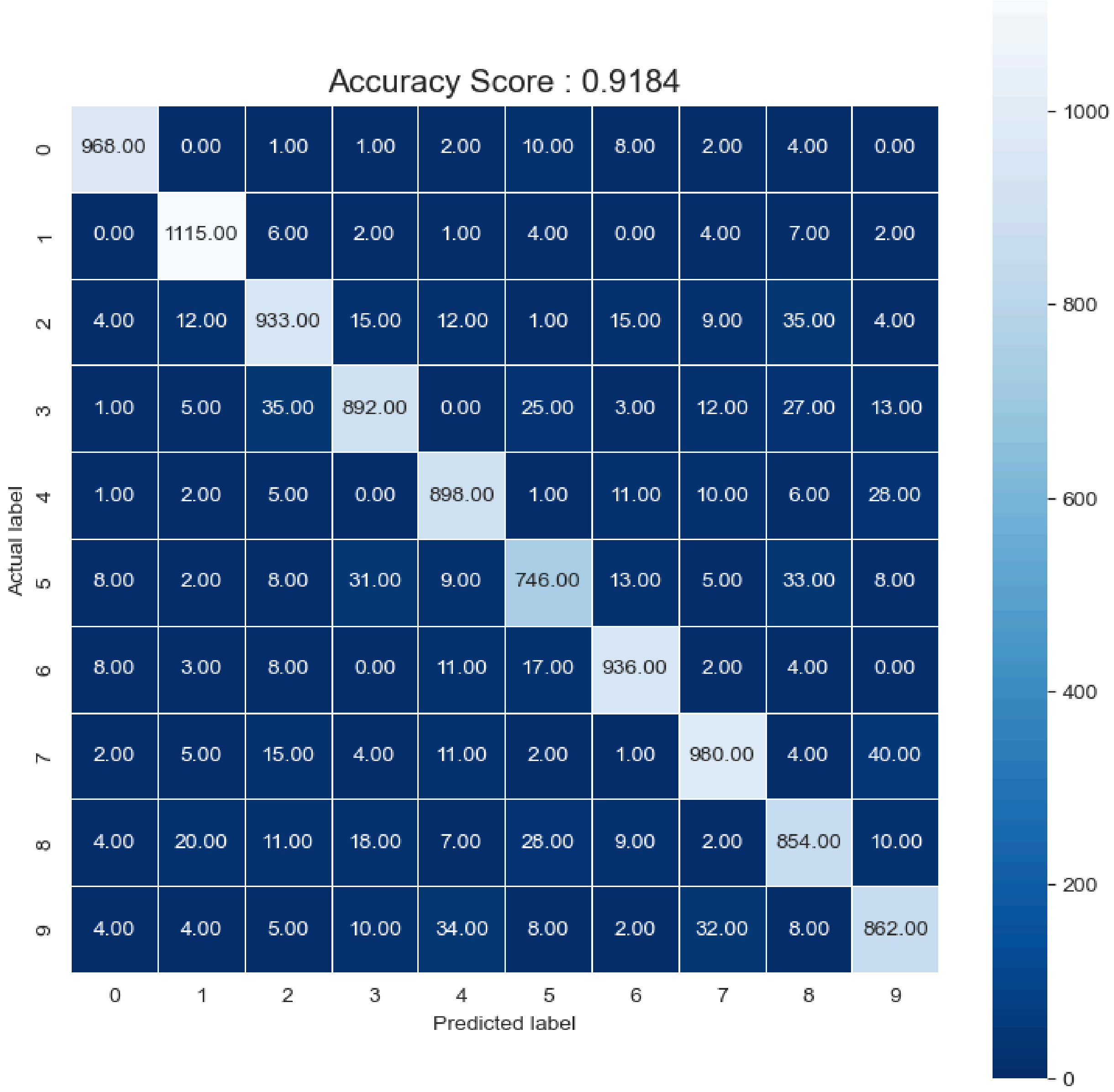
```
0.9184
```

In [33]:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, predictions)
```

In [34]:

```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt = ".2f", linewidth=0.5, square=True ,cmap ="Blues_r")
ax.xaxis.set_label_position("top")

plt.ylabel("Actual label")
plt.xlabel("Predicted label")
plt.title("Accuracy Score : {0}" .format(score), size=15)
plt.show()
```



	0	1	2	3	4	5	6	7	8	9
0	968.00	0.00	1.00	1.00	2.00	10.00	8.00	2.00	4.00	0.00
1	0.00	1115.00	6.00	2.00	1.00	4.00	0.00	4.00	7.00	2.00
2	4.00	12.00	933.00	15.00	12.00	1.00	15.00	9.00	35.00	4.00
3	1.00	5.00	35.00	892.00	0.00	25.00	3.00	12.00	27.00	13.00
4	1.00	2.00	5.00	0.00	898.00	1.00	11.00	10.00	6.00	28.00
5	8.00	2.00	8.00	31.00	9.00	746.00	13.00	5.00	33.00	8.00
6	8.00	3.00	8.00	0.00	11.00	17.00	936.00	2.00	4.00	0.00
7	2.00	5.00	15.00	4.00	11.00	2.00	1.00	980.00	4.00	40.00
8	4.00	20.00	11.00	18.00	7.00	28.00	9.00	2.00	854.00	10.00
9	4.00	4.00	5.00	10.00	34.00	8.00	2.00	32.00	8.00	862.00

Assignment on Mnist for ensemble

KNN Classifier

In [35]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

In [36]:
```
knn.fit(X,y)
```

Out[36]:
```
▾       KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

In [37]:
```
print(type(X))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [38]:

```python
X_array = X.values
```

In [39]:

```python
print(X_array.shape)
print(X_array.dtype)
```

```
(70000, 784)
float64
```

In [40]:

```python
print(X_array.flags.c_contiguous)
```

```
False
```

In [41]:

```python
X_array = np.ascontiguousarray(X_array)
```

In [42]:

```python
print(knn._fit_method)
```

```
brute
```

In [43]:

```python
y_pred = knn.predict(X_array)
```

```
C:\Users\USER\Downloads\IGAD\Lib\site-packages\sklearn\base.py:464: UserWarning: X do
es not have valid feature names, but KNeighborsClassifier was fitted with feature nam
es
  warnings.warn(
```

In [44]:

```python
print(knn.feature_names_in_)
```

```
['pixel1' 'pixel2' 'pixel3' 'pixel4' 'pixel5' 'pixel6' 'pixel7' 'pixel8'
 'pixel9' 'pixel10' 'pixel11' 'pixel12' 'pixel13' 'pixel14' 'pixel15'
 'pixel16' 'pixel17' 'pixel18' 'pixel19' 'pixel20' 'pixel21' 'pixel22'
 'pixel23' 'pixel24' 'pixel25' 'pixel26' 'pixel27' 'pixel28' 'pixel29'
 'pixel30' 'pixel31' 'pixel32' 'pixel33' 'pixel34' 'pixel35' 'pixel36'
 'pixel37' 'pixel38' 'pixel39' 'pixel40' 'pixel41' 'pixel42' 'pixel43'
 'pixel44' 'pixel45' 'pixel46' 'pixel47' 'pixel48' 'pixel49' 'pixel50'
 'pixel51' 'pixel52' 'pixel53' 'pixel54' 'pixel55' 'pixel56' 'pixel57'
 'pixel58' 'pixel59' 'pixel60' 'pixel61' 'pixel62' 'pixel63' 'pixel64'
 'pixel65' 'pixel66' 'pixel67' 'pixel68' 'pixel69' 'pixel70' 'pixel71'
 'pixel72' 'pixel73' 'pixel74' 'pixel75' 'pixel76' 'pixel77' 'pixel78'
 'pixel79' 'pixel80' 'pixel81' 'pixel82' 'pixel83' 'pixel84' 'pixel85'
 'pixel86' 'pixel87' 'pixel88' 'pixel89' 'pixel90' 'pixel91' 'pixel92'
 'pixel93' 'pixel94' 'pixel95' 'pixel96' 'pixel97' 'pixel98' 'pixel99'
 'pixel100' 'pixel101' 'pixel102' 'pixel103' 'pixel104' 'pixel105'
 'pixel106' 'pixel107' 'pixel108' 'pixel109' 'pixel110' 'pixel111'
 'pixel112' 'pixel113' 'pixel114' 'pixel115' 'pixel116' 'pixel117'
 'pixel118' 'pixel119' 'pixel120' 'pixel121' 'pixel122' 'pixel123'
 'pixel124' 'pixel125' 'pixel126' 'pixel127' 'pixel128' 'pixel129'
 'pixel130' 'pixel131' 'pixel132' 'pixel133' 'pixel134' 'pixel135'
 'pixel136' 'pixel137' 'pixel138' 'pixel139' 'pixel140' 'pixel141'
 'pixel142' 'pixel143' 'pixel144' 'pixel145' 'pixel146' 'pixel147'
 'pixel148' 'pixel149' 'pixel150' 'pixel151' 'pixel152' 'pixel153'
 'pixel154' 'pixel155' 'pixel156' 'pixel157' 'pixel158' 'pixel159'
 'pixel160' 'pixel161' 'pixel162' 'pixel163' 'pixel164' 'pixel165'
 'pixel166' 'pixel167' 'pixel168' 'pixel169' 'pixel170' 'pixel171'
 'pixel172' 'pixel173' 'pixel174' 'pixel175' 'pixel176' 'pixel177'
 'pixel178' 'pixel179' 'pixel180' 'pixel181' 'pixel182' 'pixel183'
 'pixel184' 'pixel185' 'pixel186' 'pixel187' 'pixel188' 'pixel189'
 'pixel190' 'pixel191' 'pixel192' 'pixel193' 'pixel194' 'pixel195'
 'pixel196' 'pixel197' 'pixel198' 'pixel199' 'pixel200' 'pixel201'
 'pixel202' 'pixel203' 'pixel204' 'pixel205' 'pixel206' 'pixel207'
 'pixel208' 'pixel209' 'pixel210' 'pixel211' 'pixel212' 'pixel213'
 'pixel214' 'pixel215' 'pixel216' 'pixel217' 'pixel218' 'pixel219'
 'pixel220' 'pixel221' 'pixel222' 'pixel223' 'pixel224' 'pixel225'
 'pixel226' 'pixel227' 'pixel228' 'pixel229' 'pixel230' 'pixel231'
 'pixel232' 'pixel233' 'pixel234' 'pixel235' 'pixel236' 'pixel237'
 'pixel238' 'pixel239' 'pixel240' 'pixel241' 'pixel242' 'pixel243'
 'pixel244' 'pixel245' 'pixel246' 'pixel247' 'pixel248' 'pixel249'
 'pixel250' 'pixel251' 'pixel252' 'pixel253' 'pixel254' 'pixel255'
 'pixel256' 'pixel257' 'pixel258' 'pixel259' 'pixel260' 'pixel261'
 'pixel262' 'pixel263' 'pixel264' 'pixel265' 'pixel266' 'pixel267'
 'pixel268' 'pixel269' 'pixel270' 'pixel271' 'pixel272' 'pixel273'
 'pixel274' 'pixel275' 'pixel276' 'pixel277' 'pixel278' 'pixel279'
 'pixel280' 'pixel281' 'pixel282' 'pixel283' 'pixel284' 'pixel285'
 'pixel286' 'pixel287' 'pixel288' 'pixel289' 'pixel290' 'pixel291'
 'pixel292' 'pixel293' 'pixel294' 'pixel295' 'pixel296' 'pixel297'
 'pixel298' 'pixel299' 'pixel300' 'pixel301' 'pixel302' 'pixel303'
 'pixel304' 'pixel305' 'pixel306' 'pixel307' 'pixel308' 'pixel309'
 'pixel310' 'pixel311' 'pixel312' 'pixel313' 'pixel314' 'pixel315'
 'pixel316' 'pixel317' 'pixel318' 'pixel319' 'pixel320' 'pixel321'
 'pixel322' 'pixel323' 'pixel324' 'pixel325' 'pixel326' 'pixel327'
 'pixel328' 'pixel329' 'pixel330' 'pixel331' 'pixel332' 'pixel333'
 'pixel334' 'pixel335' 'pixel336' 'pixel337' 'pixel338' 'pixel339'
 'pixel340' 'pixel341' 'pixel342' 'pixel343' 'pixel344' 'pixel345'
 'pixel346' 'pixel347' 'pixel348' 'pixel349' 'pixel350' 'pixel351'
 'pixel352' 'pixel353' 'pixel354' 'pixel355' 'pixel356' 'pixel357'
 'pixel358' 'pixel359' 'pixel360' 'pixel361' 'pixel362' 'pixel363'
 'pixel364' 'pixel365' 'pixel366' 'pixel367' 'pixel368' 'pixel369'
 'pixel370' 'pixel371' 'pixel372' 'pixel373' 'pixel374' 'pixel375'
 'pixel376' 'pixel377' 'pixel378' 'pixel379' 'pixel380' 'pixel381'
 'pixel382' 'pixel383' 'pixel384' 'pixel385' 'pixel386' 'pixel387'
 'pixel388' 'pixel389' 'pixel390' 'pixel391' 'pixel392' 'pixel393'
 'pixel394' 'pixel395' 'pixel396' 'pixel397' 'pixel398' 'pixel399'
 'pixel400' 'pixel401' 'pixel402' 'pixel403' 'pixel404' 'pixel405'
 'pixel406' 'pixel407' 'pixel408' 'pixel409' 'pixel410' 'pixel411'
 'pixel412' 'pixel413' 'pixel414' 'pixel415' 'pixel416' 'pixel417'
 'pixel418' 'pixel419' 'pixel420' 'pixel421' 'pixel422' 'pixel423'
 'pixel424' 'pixel425' 'pixel426' 'pixel427' 'pixel428' 'pixel429'
 'pixel430' 'pixel431' 'pixel432' 'pixel433' 'pixel434' 'pixel435'
 'pixel436' 'pixel437' 'pixel438' 'pixel439' 'pixel440' 'pixel441'
 'pixel442' 'pixel443' 'pixel444' 'pixel445' 'pixel446' 'pixel447'
 'pixel448' 'pixel449' 'pixel450' 'pixel451' 'pixel452' 'pixel453'
 'pixel454' 'pixel455' 'pixel456' 'pixel457' 'pixel458' 'pixel459'
 'pixel460' 'pixel461' 'pixel462' 'pixel463' 'pixel464' 'pixel465'
 'pixel466' 'pixel467' 'pixel468' 'pixel469' 'pixel470' 'pixel471'
 'pixel472' 'pixel473' 'pixel474' 'pixel475' 'pixel476' 'pixel477'
 'pixel478' 'pixel479' 'pixel480' 'pixel481' 'pixel482' 'pixel483'
 'pixel484' 'pixel485' 'pixel486' 'pixel487' 'pixel488' 'pixel489'
 'pixel490' 'pixel491' 'pixel492' 'pixel493' 'pixel494' 'pixel495'
 'pixel496' 'pixel497' 'pixel498' 'pixel499' 'pixel500' 'pixel501'
 'pixel502' 'pixel503' 'pixel504' 'pixel505' 'pixel506' 'pixel507'
 'pixel508' 'pixel509' 'pixel510' 'pixel511' 'pixel512' 'pixel513'
 'pixel514' 'pixel515' 'pixel516' 'pixel517' 'pixel518' 'pixel519'
 'pixel520' 'pixel521' 'pixel522' 'pixel523' 'pixel524' 'pixel525'
 'pixel526' 'pixel527' 'pixel528' 'pixel529' 'pixel530' 'pixel531'
 'pixel532' 'pixel533' 'pixel534' 'pixel535' 'pixel536' 'pixel537'
 'pixel538' 'pixel539' 'pixel540' 'pixel541' 'pixel542' 'pixel543'
 'pixel544' 'pixel545' 'pixel546' 'pixel547' 'pixel548' 'pixel549'
 'pixel550' 'pixel551' 'pixel552' 'pixel553' 'pixel554' 'pixel555'
 'pixel556' 'pixel557' 'pixel558' 'pixel559' 'pixel560' 'pixel561'
 'pixel562' 'pixel563' 'pixel564' 'pixel565' 'pixel566' 'pixel567'
 'pixel568' 'pixel569' 'pixel570' 'pixel571' 'pixel572' 'pixel573'
 'pixel574' 'pixel575' 'pixel576' 'pixel577' 'pixel578' 'pixel579'
 'pixel580' 'pixel581' 'pixel582' 'pixel583' 'pixel584' 'pixel585'
 'pixel586' 'pixel587' 'pixel588' 'pixel589' 'pixel590' 'pixel591'
 'pixel592' 'pixel593' 'pixel594' 'pixel595' 'pixel596' 'pixel597'
 'pixel598' 'pixel599' 'pixel600' 'pixel601' 'pixel602' 'pixel603'
 'pixel604' 'pixel605' 'pixel606' 'pixel607' 'pixel608' 'pixel609'
 'pixel610' 'pixel611' 'pixel612' 'pixel613' 'pixel614' 'pixel615'
 'pixel616' 'pixel617' 'pixel618' 'pixel619' 'pixel620' 'pixel621'
 'pixel622' 'pixel623' 'pixel624' 'pixel625' 'pixel626' 'pixel627'
 'pixel628' 'pixel629' 'pixel630' 'pixel631' 'pixel632' 'pixel633'
 'pixel634' 'pixel635' 'pixel636' 'pixel637' 'pixel638' 'pixel639'
 'pixel640' 'pixel641' 'pixel642' 'pixel643' 'pixel644' 'pixel645'
 'pixel646' 'pixel647' 'pixel648' 'pixel649' 'pixel650' 'pixel651'
 'pixel652' 'pixel653' 'pixel654' 'pixel655' 'pixel656' 'pixel657'
 'pixel658' 'pixel659' 'pixel660' 'pixel661' 'pixel662' 'pixel663'
 'pixel664' 'pixel665' 'pixel666' 'pixel667' 'pixel668' 'pixel669'
 'pixel670' 'pixel671' 'pixel672' 'pixel673' 'pixel674' 'pixel675'
 'pixel676' 'pixel677' 'pixel678' 'pixel679' 'pixel680' 'pixel681'
 'pixel682' 'pixel683' 'pixel684' 'pixel685' 'pixel686' 'pixel687'
 'pixel688' 'pixel689' 'pixel690' 'pixel691' 'pixel692' 'pixel693'
 'pixel694' 'pixel695' 'pixel696' 'pixel697' 'pixel698' 'pixel699'
 'pixel700' 'pixel701' 'pixel702' 'pixel703' 'pixel704' 'pixel705'
 'pixel706' 'pixel707' 'pixel708' 'pixel709' 'pixel710' 'pixel711'
 'pixel712' 'pixel713' 'pixel714' 'pixel715' 'pixel716' 'pixel717'
 'pixel718' 'pixel719' 'pixel720' 'pixel721' 'pixel722' 'pixel723'
 'pixel724' 'pixel725' 'pixel726' 'pixel727' 'pixel728' 'pixel729'
 'pixel730' 'pixel731' 'pixel732' 'pixel733' 'pixel734' 'pixel735'
 'pixel736' 'pixel737' 'pixel738' 'pixel739' 'pixel740' 'pixel741'
 'pixel742' 'pixel743' 'pixel744' 'pixel745' 'pixel746' 'pixel747'
 'pixel748' 'pixel749' 'pixel750' 'pixel751' 'pixel752' 'pixel753'
 'pixel754' 'pixel755' 'pixel756' 'pixel757' 'pixel758' 'pixel759'
 'pixel760' 'pixel761' 'pixel762' 'pixel763' 'pixel764' 'pixel765'
 'pixel766' 'pixel767' 'pixel768' 'pixel769' 'pixel770' 'pixel771'
 'pixel772' 'pixel773' 'pixel774' 'pixel775' 'pixel776' 'pixel777'
 'pixel778' 'pixel779' 'pixel780' 'pixel781' 'pixel782' 'pixel783'
 'pixel784']
```

In [45]:

```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
```

In [46]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y,y_pred)
```

Out[46]:

```
1.0
```

In [47]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=0, test_size= 0.
```

In [48]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

In [49]:

```
knn.fit(X_train, y_train)
```

Out[49]:

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

In [50]:

```
print(X_test)
```

```
       pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  pixel9  \
10840     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
56267     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
14849     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
62726     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
47180     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
...       ...     ...     ...     ...     ...     ...     ...     ...     ...
66702     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
12435     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
55373     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1362      0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
66135     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0

       pixel10  ...  pixel775  pixel776  pixel777  pixel778  pixel779  \
10840      0.0  ...       0.0       0.0       0.0       0.0       0.0
56267      0.0  ...       0.0       0.0       0.0       0.0       0.0
14849      0.0  ...       0.0       0.0       0.0       0.0       0.0
62726      0.0  ...       0.0       0.0       0.0       0.0       0.0
47180      0.0  ...       0.0       0.0       0.0       0.0       0.0
...        ...  ...       ...       ...       ...       ...       ...
66702      0.0  ...       0.0       0.0       0.0       0.0       0.0
12435      0.0  ...       0.0       0.0       0.0       0.0       0.0
55373      0.0  ...       0.0       0.0       0.0       0.0       0.0
1362       0.0  ...       0.0       0.0       0.0       0.0       0.0
66135      0.0  ...       0.0       0.0       0.0       0.0       0.0

       pixel780  pixel781  pixel782  pixel783  pixel784
10840       0.0       0.0       0.0       0.0       0.0
56267       0.0       0.0       0.0       0.0       0.0
14849       0.0       0.0       0.0       0.0       0.0
62726       0.0       0.0       0.0       0.0       0.0
47180       0.0       0.0       0.0       0.0       0.0
...         ...       ...       ...       ...       ...
66702       0.0       0.0       0.0       0.0       0.0
12435       0.0       0.0       0.0       0.0       0.0
55373       0.0       0.0       0.0       0.0       0.0
1362        0.0       0.0       0.0       0.0       0.0
66135       0.0       0.0       0.0       0.0       0.0

[35000 rows x 784 columns]
```

In [51]:

```
X_array_test = X_test.values
```

In [52]:

```
print(X_array_test.shape)
print(X_array_test.dtype)
```

```
(35000, 784)
float64
```

In [53]:

```
y_pred = knn.predict(X_array_test)
accuracy_score(y_test,y_pred)
```

Out[53]:

```
0.9671714285714286
```

In [54]:

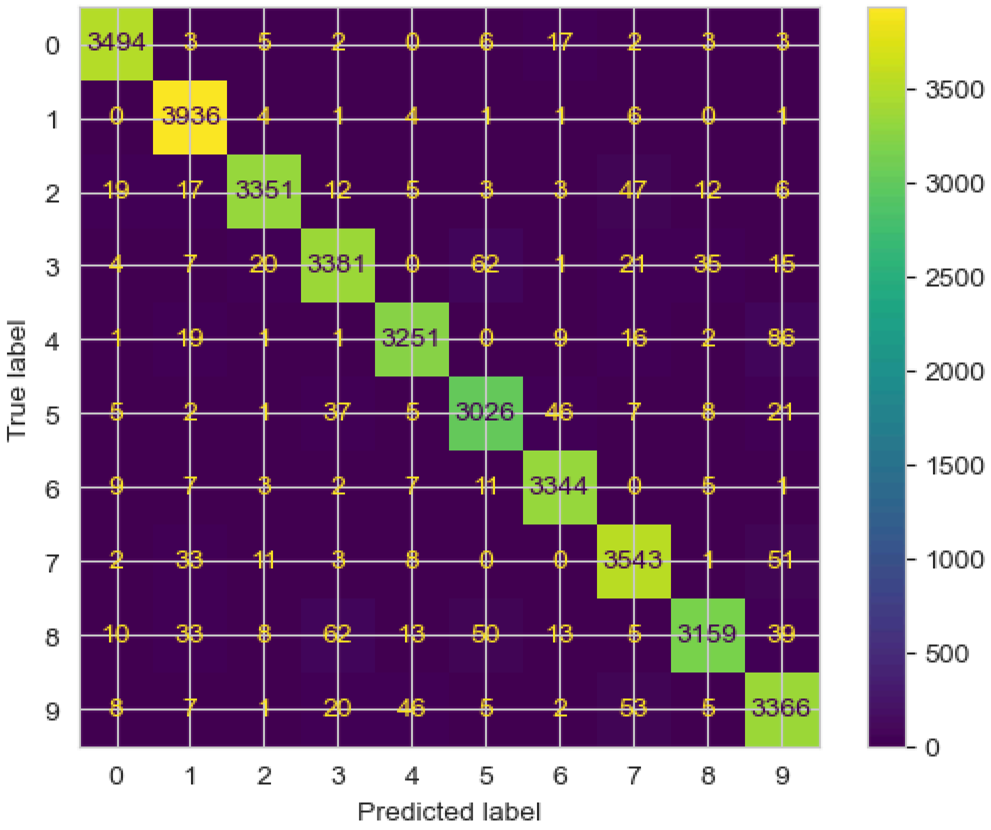
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, labels=knn.classes_.tolist()))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      3535
           1       0.97      1.00      0.98      3954
           2       0.98      0.96      0.97      3475
           3       0.96      0.95      0.96      3546
           4       0.97      0.96      0.97      3386
           5       0.96      0.96      0.96      3158
           6       0.97      0.99      0.98      3389
           7       0.96      0.97      0.96      3652
           8       0.98      0.93      0.95      3392
           9       0.94      0.96      0.95      3513

    accuracy                           0.97     35000
   macro avg       0.97      0.97      0.97     35000
weighted avg       0.97      0.97      0.97     35000
```

In [55]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
#from sklearn.metrics importclassification_report
#assuming 'knn' is your trained model, 'X_test' are your test features
predictions = knn.predict(X_array_test)
cm = confusion_matrix(y_test, predictions)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knn.classes_)
disp.plot()

plt.suptitle("Confusion Matrix for mnist Dataset")
plt.show()
```

Confusion Matrix for mnist Dataset



Decision Tree Classifier

In [56]:

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth =2, min_samples_leaf =4, random_state=42)
dt = dt.fit(X_train, y_train)
#Evaluate the model on the second set of data
y_pred = dt.predict(X_array_test)
accuracy_score(y_test, y_pred)
```

Out[56]:

```
0.34437142857142855
```

In [57]:

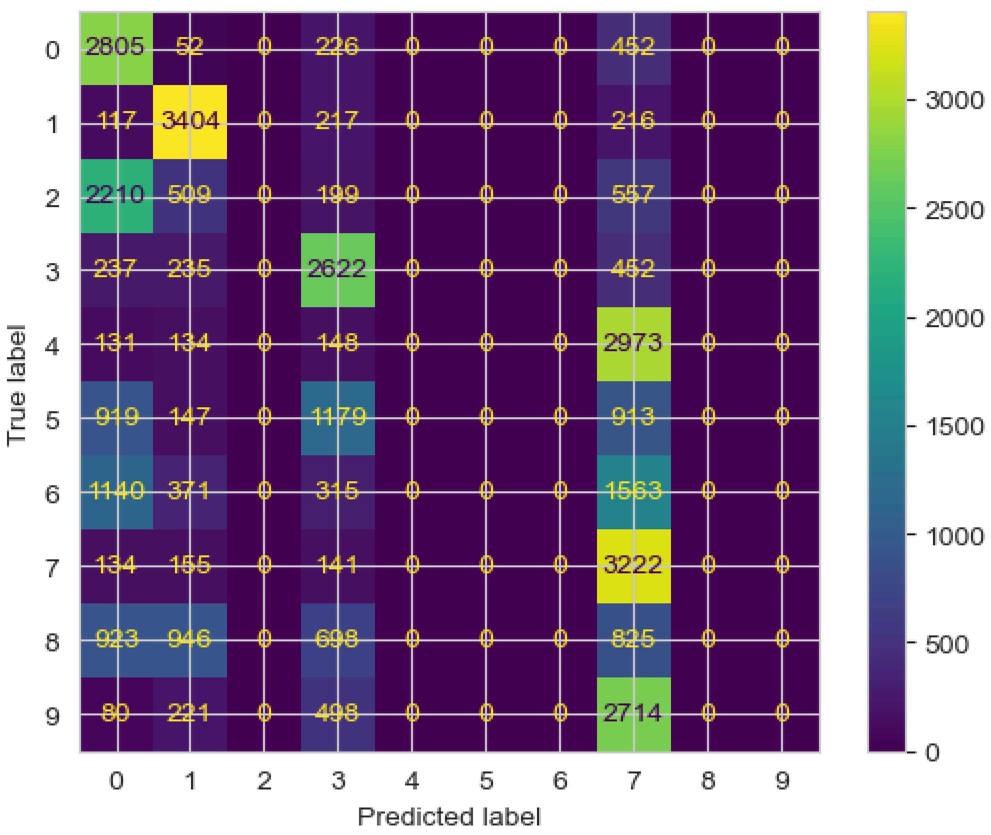
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, labels=dt.classes_.tolist()))
```

```
              precision    recall  f1-score   support

           0       0.32      0.79      0.46      3535
           1       0.55      0.86      0.67      3954
           2       0.00      0.00      0.00      3475
           3       0.42      0.74      0.54      3546
           4       0.00      0.00      0.00      3386
           5       0.00      0.00      0.00      3158
           6       0.00      0.00      0.00      3389
           7       0.23      0.88      0.37      3652
           8       0.00      0.00      0.00      3392
           9       0.00      0.00      0.00      3513

    accuracy                           0.34     35000
   macro avg       0.15      0.33      0.20     35000
weighted avg       0.16      0.34      0.21     35000
```

In [58]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

#assuming 'Decision Tree' is your trained model, 'X_test' are your test features
predictions = dt.predict(X_array_test)
cm = confusion_matrix(y_test, predictions)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=dt.classes_)
disp.plot()

plt.suptitle("Confusion Matrix for mnist Dataset")
plt.show()
```

Confusion Matrix for mnist Dataset



	0	1	2	3	4	5	6	7	8	9
0	2805	52	0	226	0	0	0	452	0	0
1	117	3404	0	217	0	0	0	216	0	0
2	2210	509	0	199	0	0	0	557	0	0
3	237	235	0	2622	0	0	0	452	0	0
4	131	134	0	148	0	0	0	2973	0	0
5	919	147	0	1179	0	0	0	913	0	0
6	1140	371	0	315	0	0	0	1563	0	0
7	134	155	0	141	0	0	0	3222	0	0
8	923	946	0	698	0	0	0	825	0	0
9	80	221	0	498	0	0	0	2714	0	0

Bagging Classifeir

In [59]:
```
#Bagging Algorithm
from sklearn.ensemble import BaggingClassifier
BaggingClassifier?
```

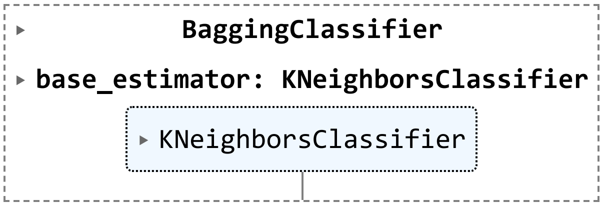
In [60]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=3)
bag = BaggingClassifier(knn,
                        max_samples= .5, max_features=28)
```

In [61]:
```
bag.fit(X_train, y_train)
```

Out[61]:
```
▸        BaggingClassifier
▸ estimator: KNeighborsClassifier
      ▸ KNeighborsClassifier
```

In [62]:
```
BaggingClassifier(base_estimator=KNeighborsClassifier(n_neighbors=3),
                  max_features=2, max_samples=0.5, n_jobs=2, oob_score=True)
```

Out[62]:



In [63]:

```python
#evaluate the model
y_pred= bag.predict(X_array_test)
accuracy_score(y_test, y_pred)
```

Out[63]:

```
0.794
```

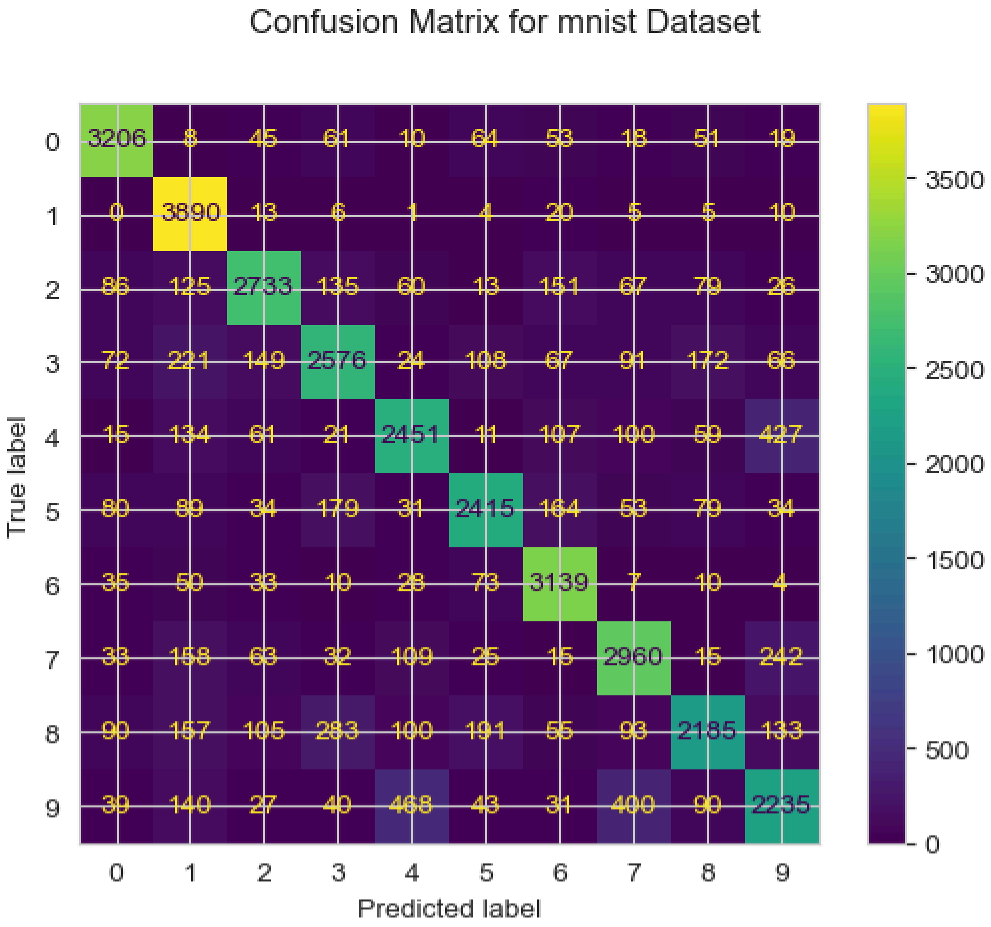
In [64]:

```python
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, labels=bag.classes_.tolist()))
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.89      3535
           1       0.78      0.98      0.87      3954
           2       0.84      0.79      0.81      3475
           3       0.77      0.73      0.75      3546
           4       0.75      0.72      0.74      3386
           5       0.82      0.76      0.79      3158
           6       0.83      0.93      0.87      3389
           7       0.78      0.81      0.80      3652
           8       0.80      0.64      0.71      3392
           9       0.70      0.64      0.67      3513

    accuracy                           0.79     35000
   macro avg       0.79      0.79      0.79     35000
weighted avg       0.79      0.79      0.79     35000
```

In [65]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

#assuming 'bagging classfier' is your trained model, 'X_test' are your test features
predictions = bag.predict(X_array_test)
cm = confusion_matrix(y_test, predictions)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=bag.classes_)
disp.plot()

plt.suptitle("Confusion Matrix for mnist Dataset")
plt.show()
```



Random Forest Classifier

In [66]:
```
from sklearn.ensemble import RandomForestClassifier
RandomForestClassifier?
```

In [67]:
```
rf = RandomForestClassifier(n_estimators=20)
```

In [68]:
```
rf.fit(X_train,y_train)
```

Out[68]:
```
▾      RandomForestClassifier
RandomForestClassifier(n_estimators=20)
```

In [69]:
```
#Evaluate the model
y_pred = rf.predict(X_array_test)
accuracy_score(y_test, y_pred)
```

Out[69]:
```
0.9517142857142857
```

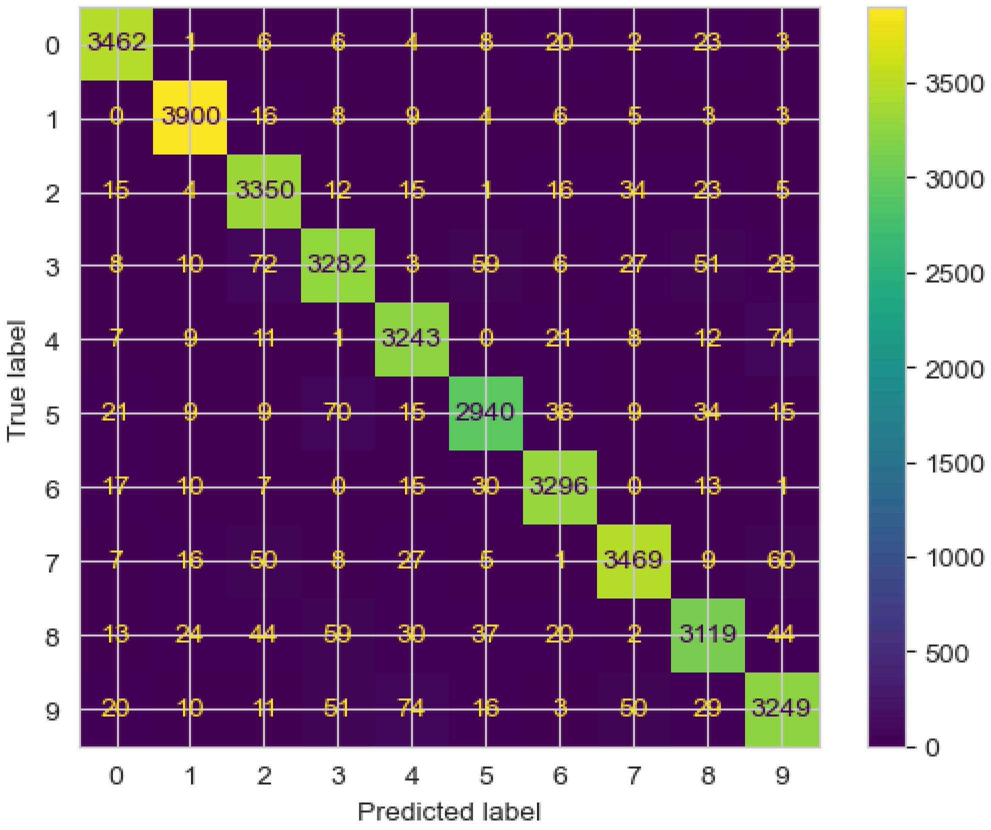
In [70]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, labels=rf.classes_.tolist()))
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      3535
           1       0.98      0.99      0.98      3954
           2       0.94      0.96      0.95      3475
           3       0.94      0.93      0.93      3546
           4       0.94      0.96      0.95      3386
           5       0.95      0.93      0.94      3158
           6       0.96      0.97      0.97      3389
           7       0.96      0.95      0.96      3652
           8       0.94      0.92      0.93      3392
           9       0.93      0.92      0.93      3513

    accuracy                           0.95     35000
   macro avg       0.95      0.95      0.95     35000
weighted avg       0.95      0.95      0.95     35000
```

In [71]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

#assuming 'rf' is your trained model, 'X_test' are your test features
predictions = rf.predict(X_array_test)
cm = confusion_matrix(y_test, predictions)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rf.classes_)
disp.plot()

plt.suptitle("Confusion Matrix for mnist Dataset")
plt.show()
```

Confusion Matrix for mnist Dataset



	0	1	2	3	4	5	6	7	8	9
0	3462	1	6	6	4	8	20	2	23	3
1	0	3900	16	8	9	4	6	5	3	3
2	15	4	3350	12	15	1	16	34	23	5
3	8	10	72	3282	3	59	6	27	51	28
4	7	9	11	1	3243	0	21	8	12	74
5	21	9	9	70	15	2940	36	9	34	15
6	17	10	7	0	15	30	3296	0	13	1
7	7	16	50	8	27	5	1	3469	9	60
8	13	24	44	59	30	37	20	2	3119	44
9	20	10	11	51	74	16	3	50	29	3249

AdaBoost Classifier

In [72]:
```
from sklearn.ensemble import AdaBoostClassifier
AdaBoostClassifier?
```

In [73]:
```
ada=AdaBoostClassifier(n_estimators=100)
```

In [74]:
```
ada.fit(X_train, y_train)
```

Out[74]:
```
▾        AdaBoostClassifier
AdaBoostClassifier(n_estimators=100)
```

In [75]:
```
#evaluate the model
y_pred = ada.predict(X_array_test)
accuracy_score(y_test, y_pred)
```

Out[75]:
```
0.6880857142857143
```

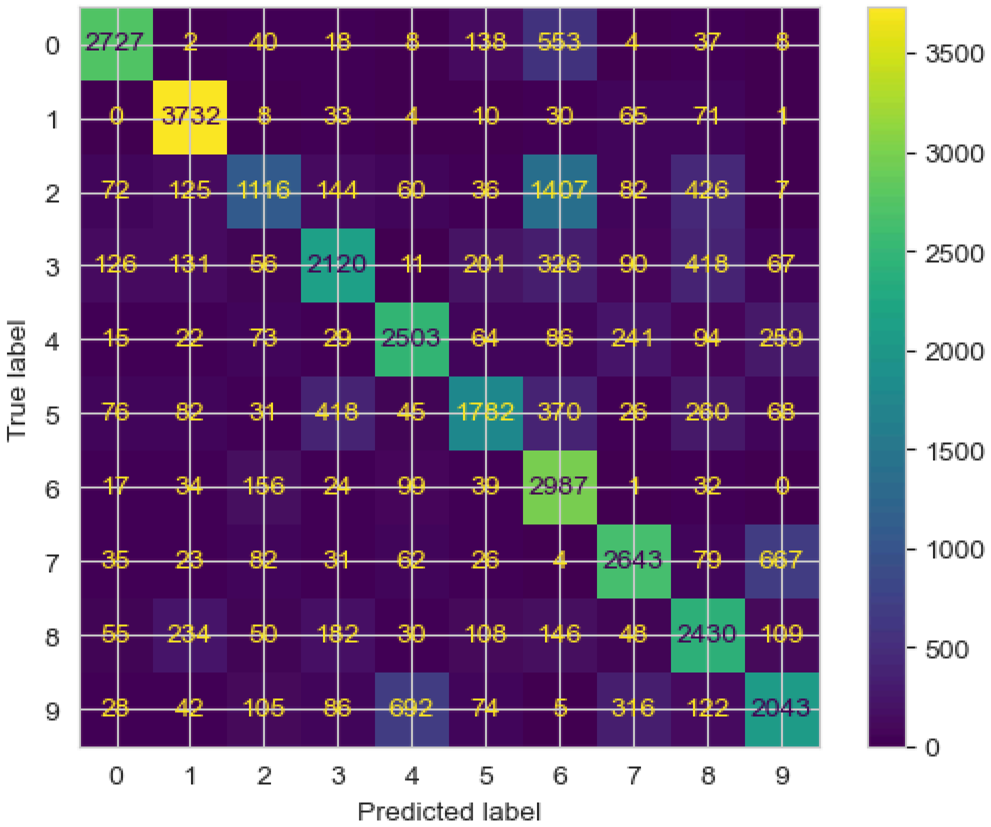
In [76]:
```
 from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, labels=ada.classes_.tolist()))
```

```
              precision    recall  f1-score   support

           0       0.87      0.77      0.82      3535
           1       0.84      0.94      0.89      3954
           2       0.65      0.32      0.43      3475
           3       0.69      0.60      0.64      3546
           4       0.71      0.74      0.73      3386
           5       0.72      0.56      0.63      3158
           6       0.51      0.88      0.64      3389
           7       0.75      0.72      0.74      3652
           8       0.61      0.72      0.66      3392
           9       0.63      0.58      0.61      3513

    accuracy                           0.69     35000
   macro avg       0.70      0.68      0.68     35000
weighted avg       0.70      0.69      0.68     35000
```

In [77]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

#assuming 'adaboost' is your trained model, 'X_test' are your test features
predictions = ada.predict(X_array_test)
cm = confusion_matrix(y_test, predictions)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=ada.classes_)
disp.plot()

plt.suptitle("Confusion Matrix for mnist Dataset")
plt.show()
```

Confusion Matrix for mnist Dataset



```
In [ ]:
```