1.Importing Libraries:

In [413...]
```
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [414...]
```
%matplotlib inline
```

In [415...]
```
pd.set_option("display.max_columns", 100)
```

2. Data Loading:

In [416...]
```
df = pd.read_csv("bank marketing v2.csv")
df.head()
```

Out[416...]

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duratic
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	10
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	14
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	13
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	5
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	6

3. Data Checks:

In [417...]
```
df.shape
```

Out[417...]
```
(11162, 15)
```

In [418...]
```
df.dtypes
```

Out[418...]
```
age           int64
job          object
marital      object
education    object
default      object
balance       int64
housing      object
loan         object
contact      object
day           int64
month        object
duration      int64
poutcome     object
deposit      object
```

```
p_recency    object
dtype: object
```

In [419…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11162 entries, 0 to 11161
Data columns (total 15 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        11162 non-null  int64 
 1   job        11162 non-null  object
 2   marital    11162 non-null  object
 3   education  11162 non-null  object
 4   default    11162 non-null  object
 5   balance    11162 non-null  int64 
 6   housing    11162 non-null  object
 7   loan       11162 non-null  object
 8   contact    11162 non-null  object
 9   day        11162 non-null  int64 
 10  month      11162 non-null  object
 11  duration   11162 non-null  int64 
 12  poutcome   11162 non-null  object
 13  deposit    11162 non-null  object
 14  p_recency  11162 non-null  object
dtypes: int64(4), object(11)
memory usage: 1.3+ MB
```

In [420…

```
df.isnull().sum()
```

Out[420…

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
poutcome     0
deposit      0
p_recency    0
dtype: int64
```

In [421…

```
# Checking Categorical Features:

cv = ["job","marital","education","default","housing","loan","contact","month","poutcom

for i in cv:
    print(df[i].value_counts())
    print("-"*50)
```

```
management       2566
blue-collar      1944
technician       1823
```

```
admin.           1334
services          923
retired           778
self-employed     405
student           360
unemployed        357
entrepreneur      328
housemaid         274
unknown            70
Name: job, dtype: int64
-------------------------------------------------
married     6351
single      3518
divorced    1293
Name: marital, dtype: int64
-------------------------------------------------
secondary    5476
tertiary     3689
primary      1500
unknown       497
Name: education, dtype: int64
-------------------------------------------------
no     10994
yes      168
Name: default, dtype: int64
-------------------------------------------------
no     5881
yes    5281
Name: housing, dtype: int64
-------------------------------------------------
no     9702
yes    1460
Name: loan, dtype: int64
-------------------------------------------------
cellular     8042
unknown      2346
telephone     774
Name: contact, dtype: int64
-------------------------------------------------
may    2824
aug    1519
jul    1514
jun    1222
nov     943
apr     923
feb     776
oct     392
jan     344
sep     319
mar     276
dec     110
Name: month, dtype: int64
-------------------------------------------------
unknown    8326
failure    1228
success    1071
other       537
Name: poutcome, dtype: int64
-------------------------------------------------
no     5873
```

```
yes    5289
Name: deposit, dtype: int64
------------------------------------------------
None      8324
b_1yr     1307
a_6m      1279
c_1yr+     252
Name: p_recency, dtype: int64
------------------------------------------------
```

4. Exploratory Data Analysis:

In [422…

```
# Binary mapping for Categorical Features:
df["default"] = df["default"].map({"yes":1,"no":0})
df["housing"] = df["housing"].map({"yes":1,"no":0})
df["loan"] = df["loan"].map({"yes":1,"no":0})
df["deposit"] = df["deposit"].map({"yes":1,"no":0})
```

In [423…

```
df.head()
```

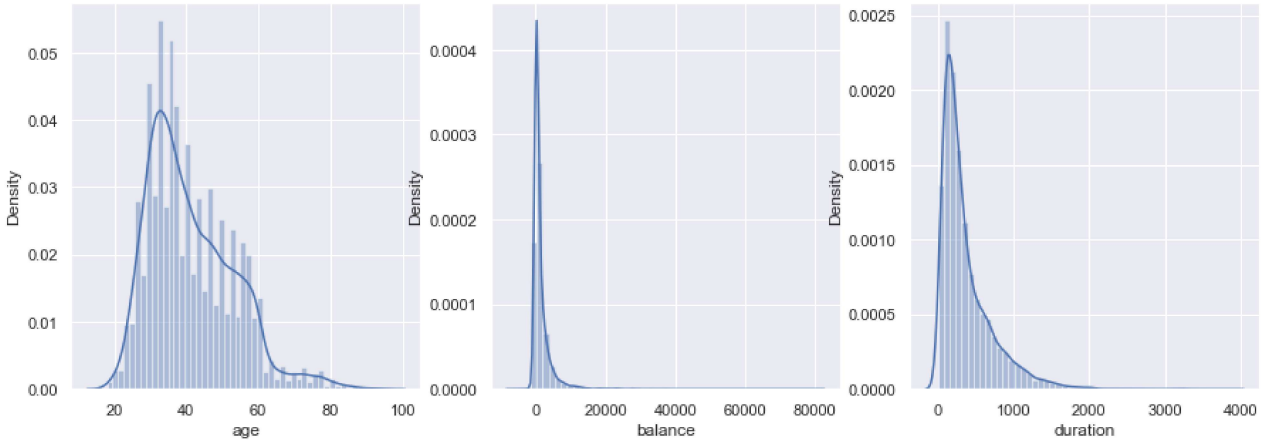
Out[423…

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duratic
0	59	admin.	married	secondary	0	2343	1	0	unknown	5	may	10
1	56	admin.	married	secondary	0	45	0	0	unknown	5	may	14
2	41	technician	married	secondary	0	1270	1	0	unknown	5	may	13
3	55	services	married	secondary	0	2476	1	0	unknown	5	may	5
4	54	admin.	married	tertiary	0	184	0	0	unknown	5	may	6

- Univariate Analysis:

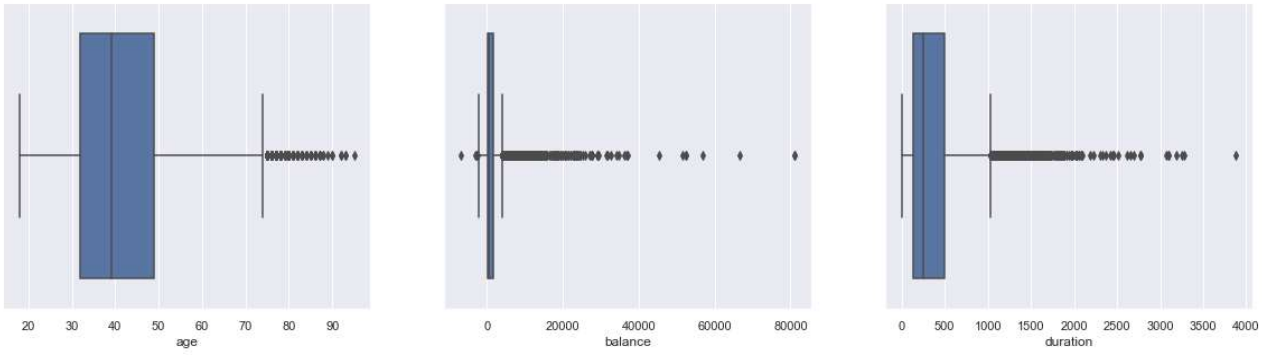
In [424…

```
# Distribution plots - Numerical Features:

plt.figure(figsize=(15,5))
sns.set_theme()

plt.subplot(1,3,1)
sns.distplot(df["age"])

plt.subplot(1,3,2)
sns.distplot(df["balance"])

plt.subplot(1,3,3)
sns.distplot(df["duration"])

plt.show()
```



In [425…

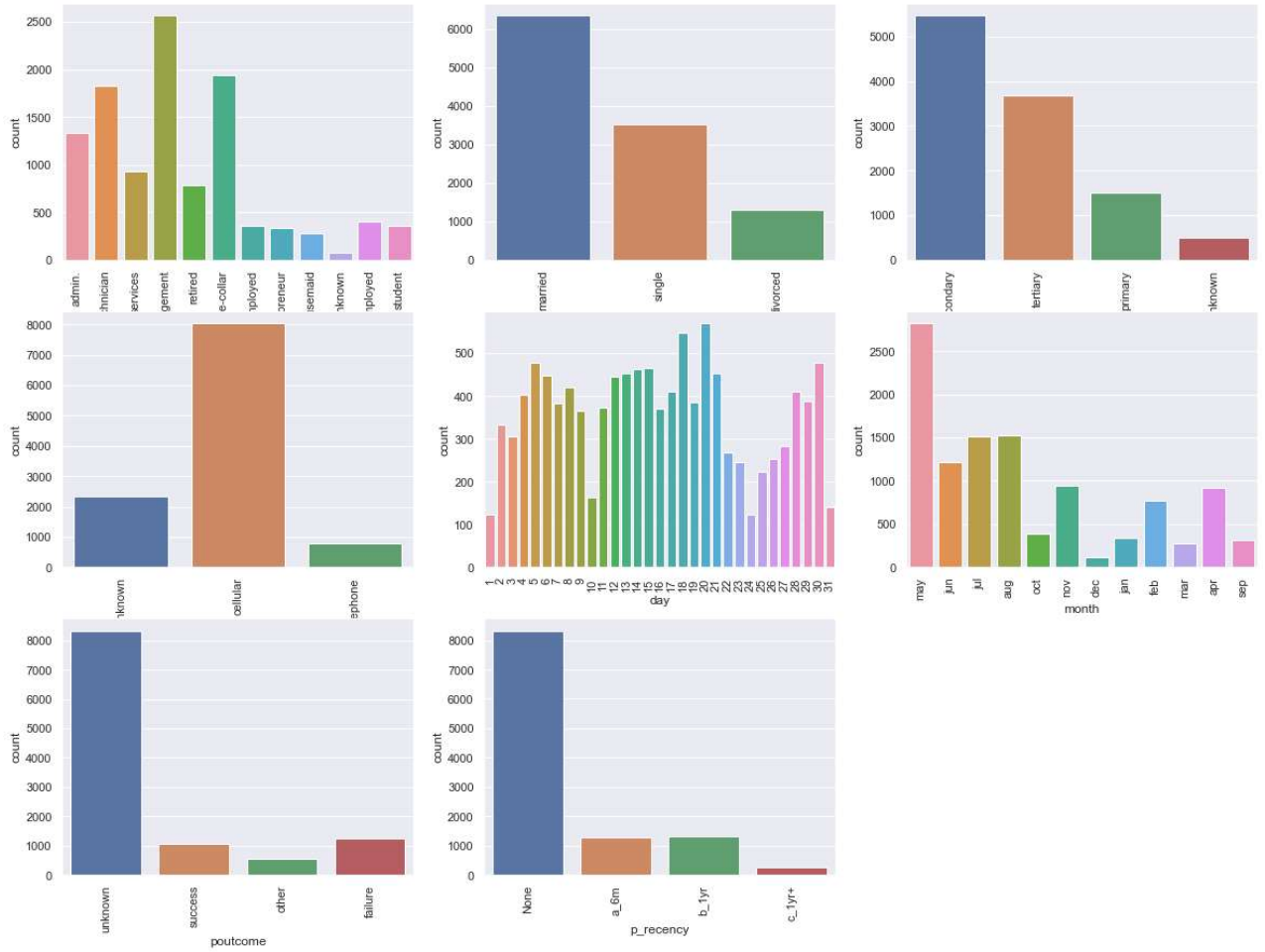
```
# Box plots - Numerical Features:

plt.figure(figsize=(20,5))
sns.set_theme()

plt.subplot(1,3,1)
sns.boxplot(df["age"])

plt.subplot(1,3,2)
sns.boxplot(df["balance"])

plt.subplot(1,3,3)
sns.boxplot(df["duration"])

plt.show()
```



In [426…

```
# Categorical Features:
plt.figure(figsize=(20,15))

plt.subplot(3,3,1)
sns.countplot(df["job"])
plt.xticks(rotation = 90)

plt.subplot(3,3,2)
sns.countplot(df["marital"])
plt.xticks(rotation = 90)

plt.subplot(3,3,3)
sns.countplot(df["education"])
plt.xticks(rotation = 90)
```

```
plt.subplot(3,3,4)
sns.countplot(df["contact"])
plt.xticks(rotation = 90)

plt.subplot(3,3,5)
sns.countplot(df["day"])
plt.xticks(rotation = 90)

plt.subplot(3,3,6)
sns.countplot(df["month"])
plt.xticks(rotation = 90)

plt.subplot(3,3,7)
sns.countplot(df["poutcome"])
plt.xticks(rotation = 90)


plt.subplot(3,3,8)
sns.countplot(df["p_recency"])
plt.xticks(rotation = 90)

plt.show()
```



In [427…

```
df.head()
```

Out[427…

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duratic
0	59	admin.	married	secondary	0	2343	1	0	unknown	5	may	104

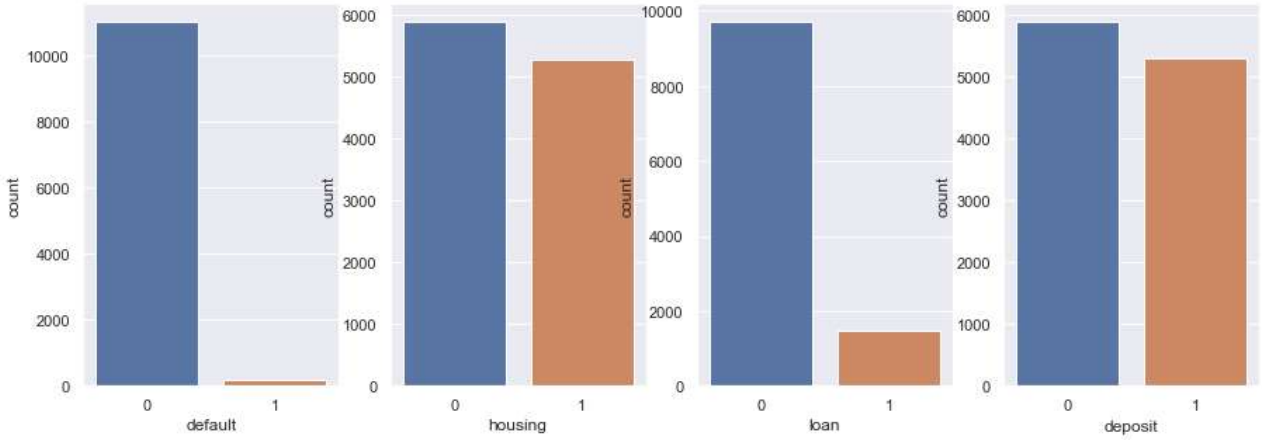
	age	job	marital	education	default	balance	housing	loan	contact	day	month	duratic
1	56	admin.	married	secondary	0	45	0	0	unknown	5	may	146
2	41	technician	married	secondary	0	1270	1	0	unknown	5	may	138
3	55	services	married	secondary	0	2476	1	0	unknown	5	may	57
4	54	admin.	married	tertiary	0	184	0	0	unknown	5	may	67

In [428…

```
# Binary Categorical Variables:

plt.figure(figsize=(15,5))

plt.subplot(1,4,1)
sns.countplot(df["default"])
plt.subplot(1,4,2)
sns.countplot(df["housing"])
plt.subplot(1,4,3)
sns.countplot(df["loan"])
plt.subplot(1,4,4)
sns.countplot(df["deposit"])
```

Out[428…

```
<AxesSubplot:xlabel='deposit', ylabel='count'>
```



In [429…

```
# Class Imbalance:
df["deposit"].value_counts(normalize=True)*100
```

Out[429…

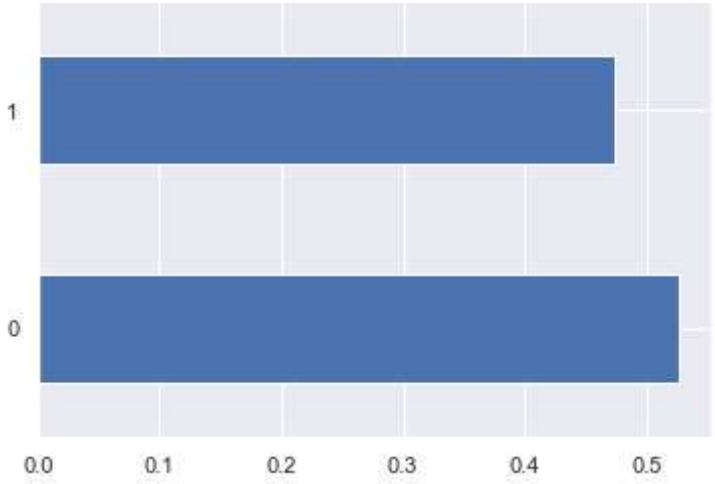
```
0    52.616019
1    47.383981
Name: deposit, dtype: float64
```

In [430…

```
df["deposit"].value_counts(normalize=True).plot.barh()
```

Out[430…

```
<AxesSubplot:>
```



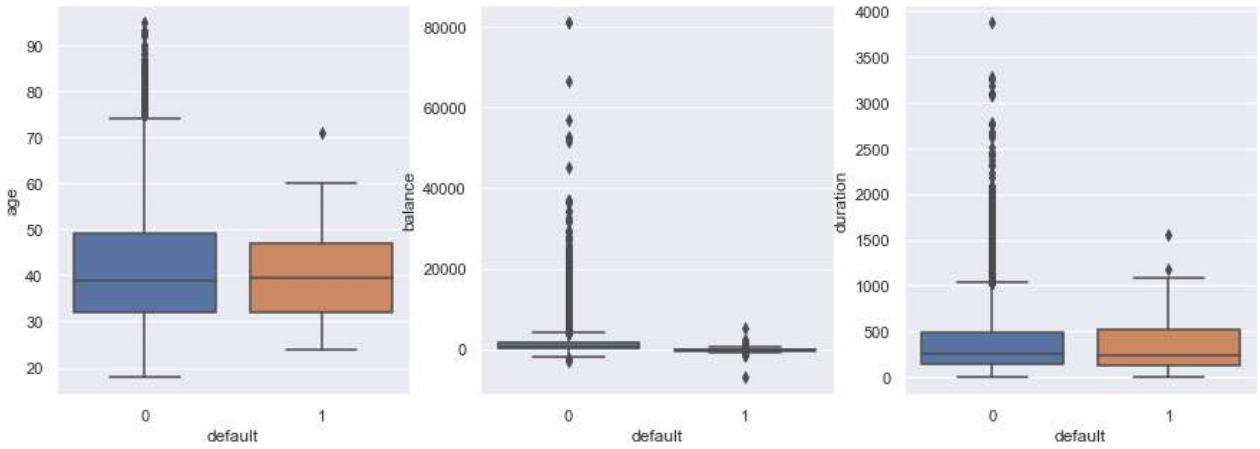
- Bi- Variate Analysis:

In [431…

```
plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
sns.boxplot(x=df["default"], y=df["age"], data=df)
plt.subplot(1,3,2)
sns.boxplot(x=df["default"], y=df["balance"], data=df)
plt.subplot(1,3,3)
sns.boxplot(x=df["default"], y=df["duration"], data=df)
```

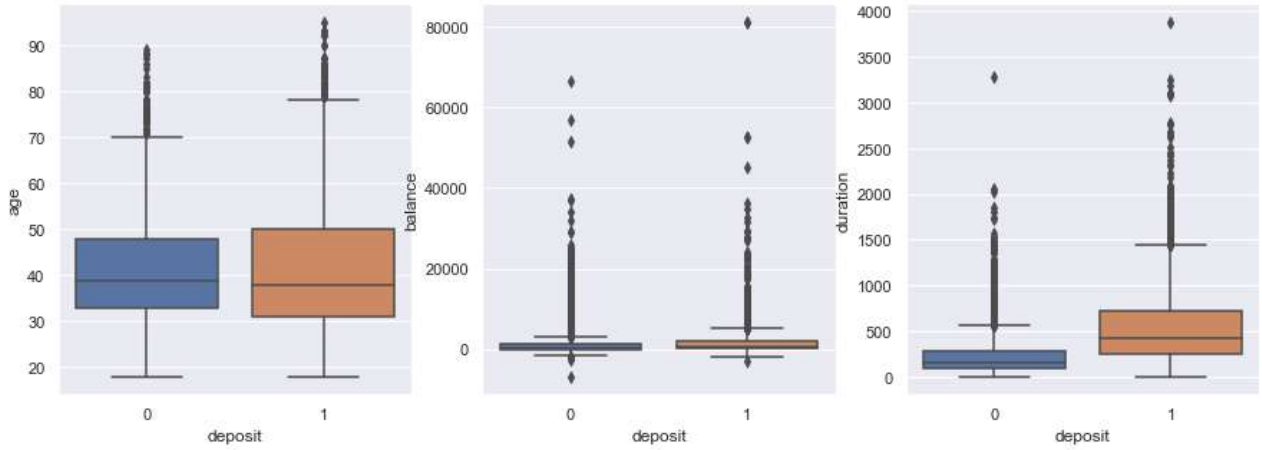
Out[431…

```
<AxesSubplot:xlabel='default', ylabel='duration'>
```



In [432…

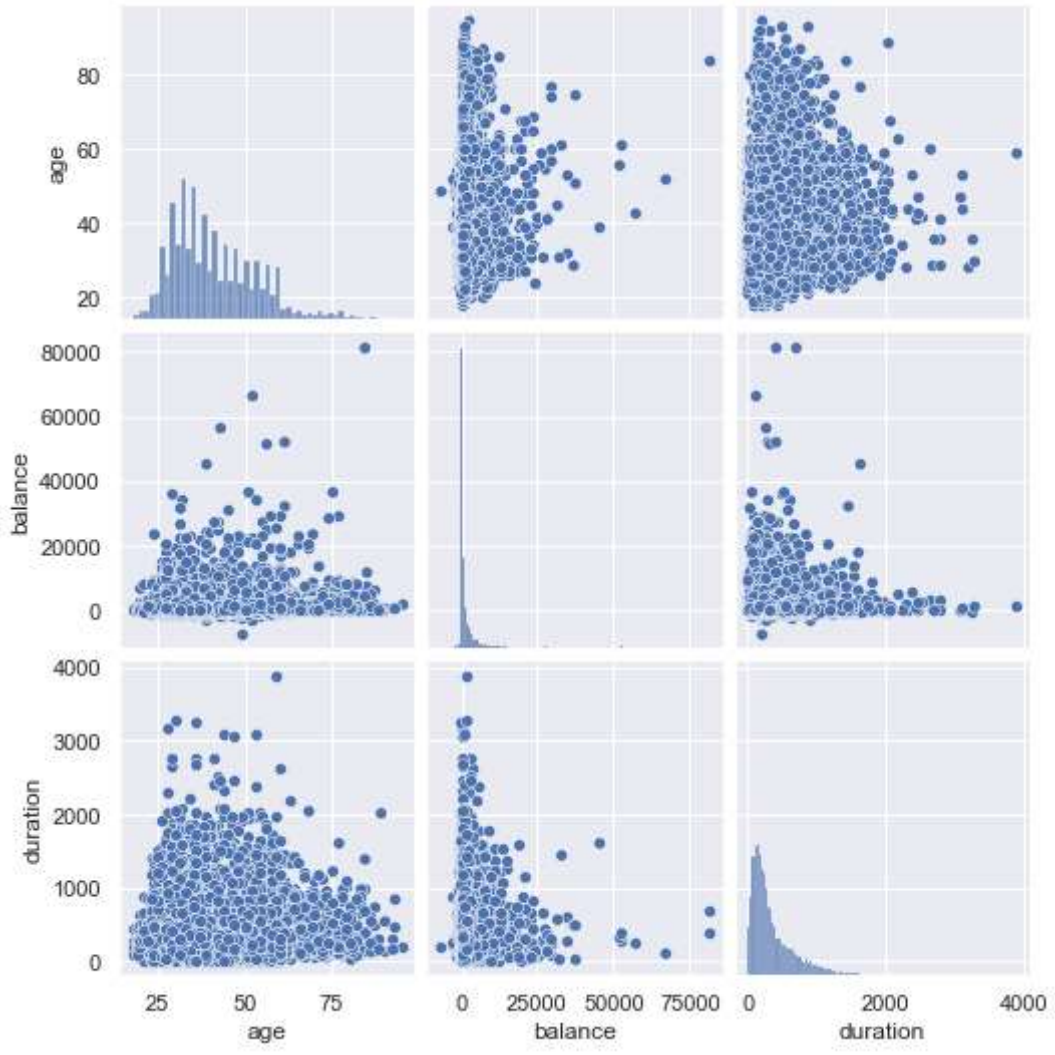
```
plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
sns.boxplot(x=df["deposit"], y=df["age"], data=df)
plt.subplot(1,3,2)
sns.boxplot(x=df["deposit"], y=df["balance"], data=df)
plt.subplot(1,3,3)
sns.boxplot(x=df["deposit"], y=df["duration"], data=df)

plt.show()
```



In [433…

```
plt.figure(figsize=(15,3))
sns.pairplot(df[["age","balance","duration"]])
plt.show()
```

```
<Figure size 1080x216 with 0 Axes>
```



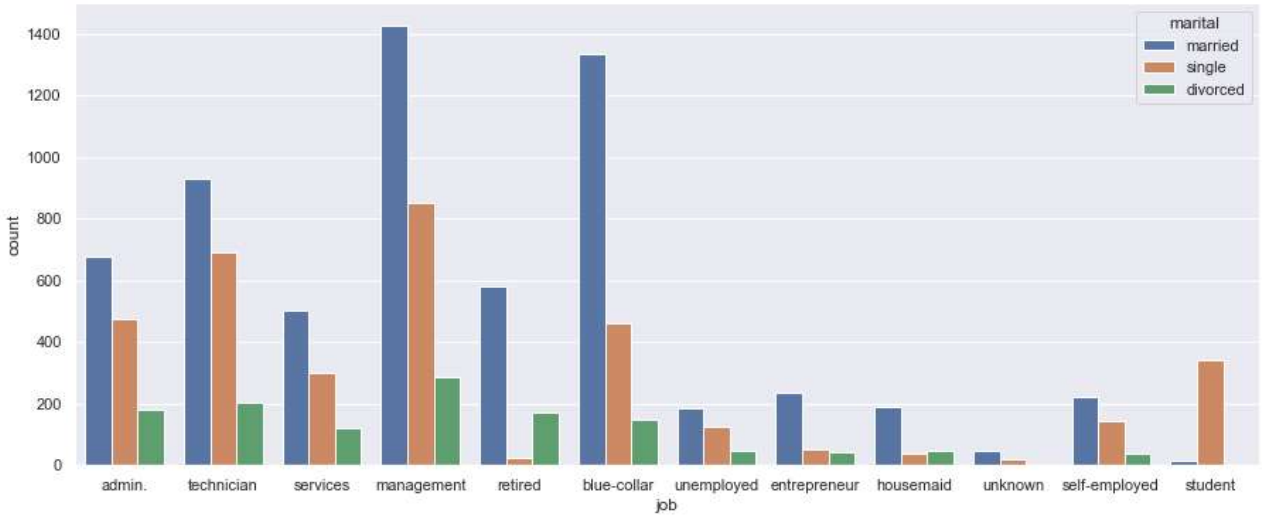
In [434…

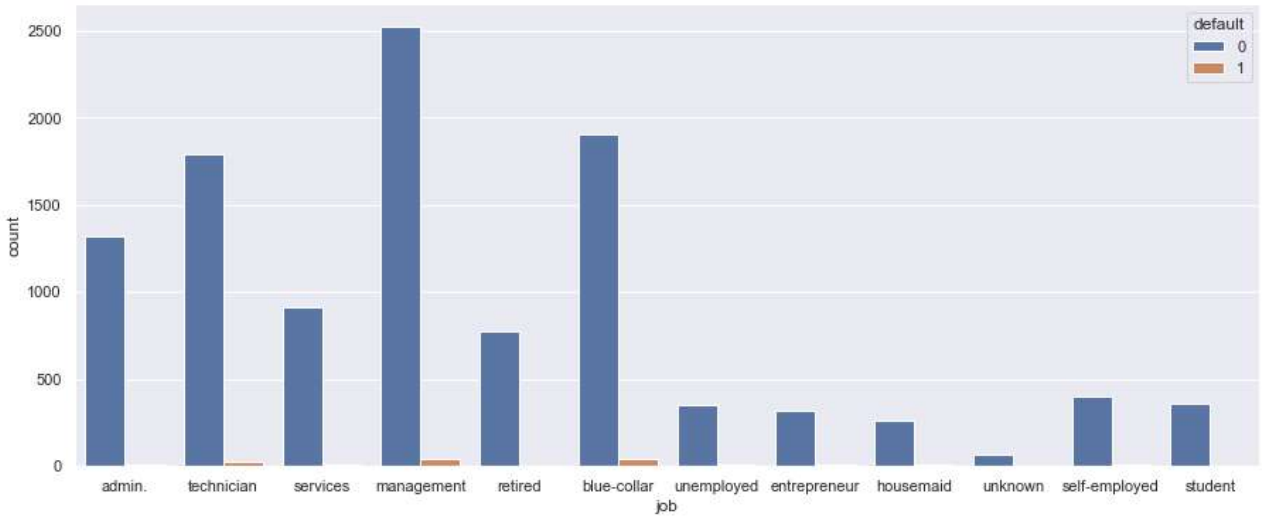
```
plt.figure(figsize=(15,6))
sns.countplot(x=df["job"],hue=df["marital"], data=df)

plt.figure(figsize=(15,6))
```

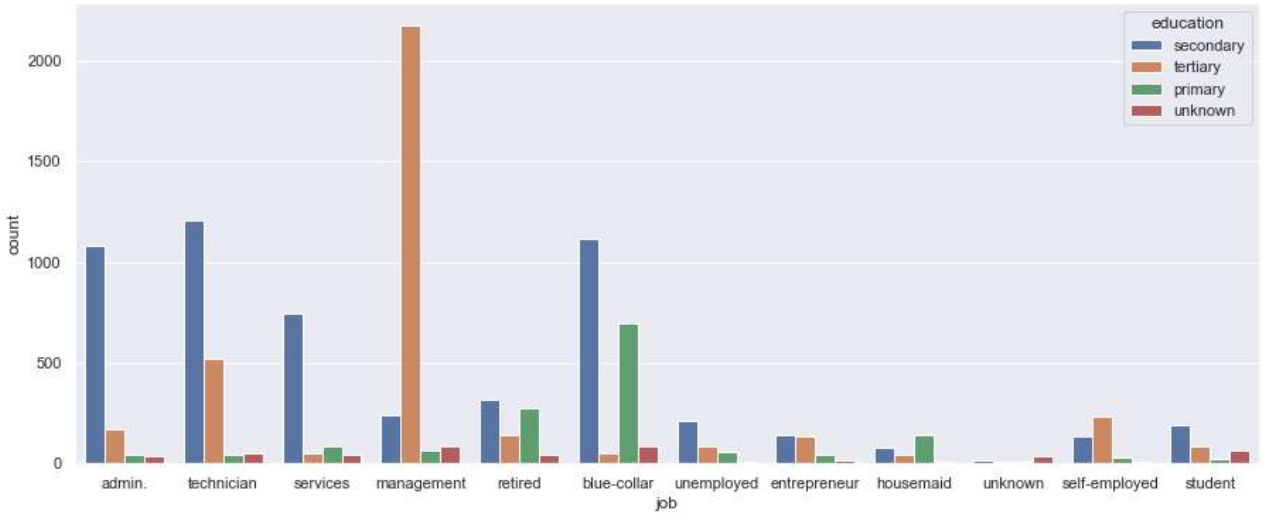
```
sns.countplot(x=df["job"],hue=df["default"], data=df)

plt.figure(figsize=(15,6))
sns.countplot(x=df["job"],hue=df["education"], data=df)
```

Out[434…

```
<AxesSubplot:xlabel='job', ylabel='count'>
```





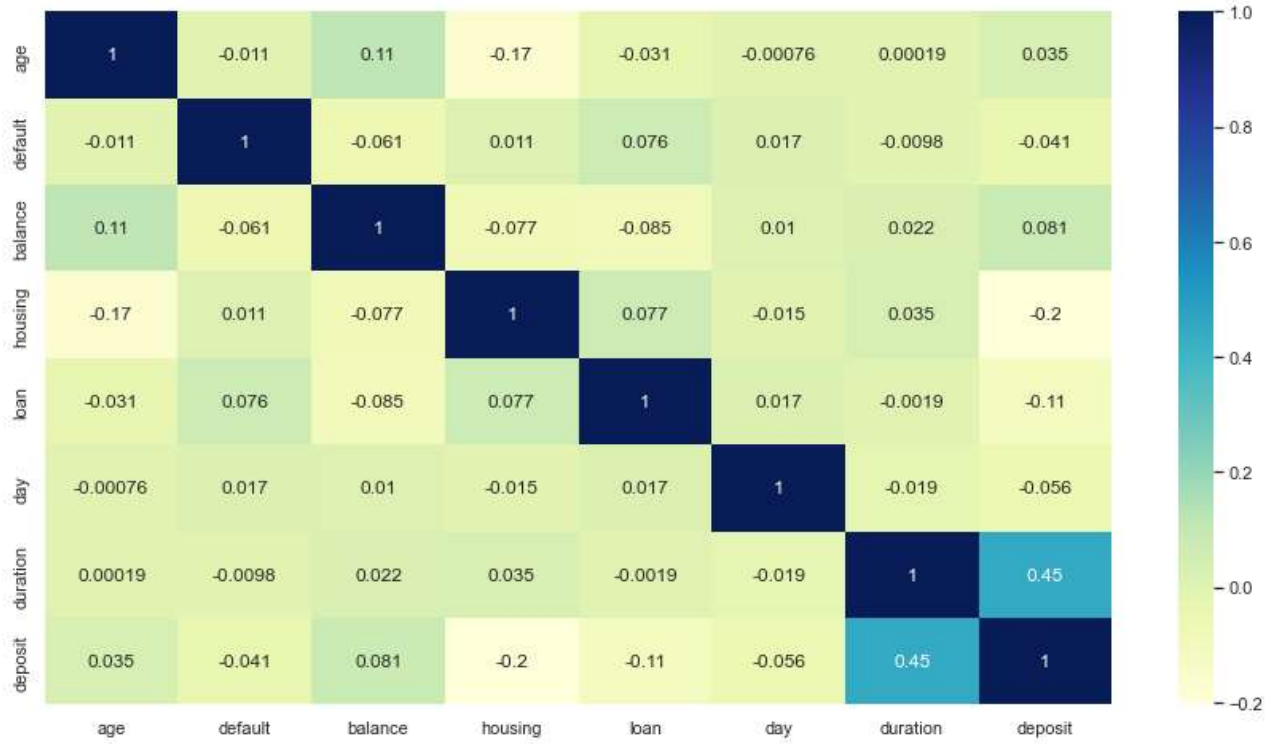


In [435…

```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(), annot=True,cmap="YlGnBu")
```

Out[435…

```
<AxesSubplot:>
```



5. Data Preparation for model Building:

In [436…

```
# Dropping Day and Duration Features:

df = df.drop(["day","duration"],axis=1)
```

In [437…

```
df.head()
```

Out[437…

	age	job	marital	education	default	balance	housing	loan	contact	month	poutcome
0	59	admin.	married	secondary	0	2343	1	0	unknown	may	unknown
1	56	admin.	married	secondary	0	45	0	0	unknown	may	unknown
2	41	technician	married	secondary	0	1270	1	0	unknown	may	unknown
3	55	services	married	secondary	0	2476	1	0	unknown	may	unknown
4	54	admin.	married	tertiary	0	184	0	0	unknown	may	unknown

In [438…

```
# Creating Dummy Features for job, marital, education, contact, month, poutcome,  p_rec

df_dummies = ["job", "marital","education","contact","month","poutcome","p_recency"]
```

In [439…

```
df_dummy_frames = pd.get_dummies(df[df_dummies], drop_first=True)
df_dummy_frames.head()
```

Out[439…

	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	j
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	1	
4	0	0	0	0	0	0	0	

In [440…

```
df_dummy_frames.shape
```

Out[440…

```
(11162, 35)
```

In [441…

```
# Dropping Categorical features:
df.drop(df_dummies, axis=1, inplace=True)
df.head()
```

Out[441…

	age	default	balance	housing	loan	deposit
0	59	0	2343	1	0	1
1	56	0	45	0	0	1
2	41	0	1270	1	0	1
3	55	0	2476	1	0	1
4	54	0	184	0	0	1

In [442…

```
# Adding the Data frames:
df = pd.concat([df,df_dummy_frames], axis=1)
df.head()
```

Out[442…

	age	default	balance	housing	loan	deposit	job_blue-collar	job_entrepreneur	job_housemaid	job_mar
0	59	0	2343	1	0	1	0	0	0	
1	56	0	45	0	0	1	0	0	0	
2	41	0	1270	1	0	1	0	0	0	
3	55	0	2476	1	0	1	0	0	0	
4	54	0	184	0	0	1	0	0	0	

In [443…

```
df.shape
```

Out[443…

```
(11162, 41)
```

In [444...
```
df.columns
```

Out[444...
```
Index(['age', 'default', 'balance', 'housing', 'loan', 'deposit',
       'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 'job_services',
       'job_student', 'job_technician', 'job_unemployed', 'job_unknown',
       'marital_married', 'marital_single', 'education_secondary',
       'education_tertiary', 'education_unknown', 'contact_telephone',
       'contact_unknown', 'month_aug', 'month_dec', 'month_feb', 'month_jan',
       'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov',
       'month_oct', 'month_sep', 'poutcome_other', 'poutcome_success',
       'poutcome_unknown', 'p_recency_a_6m', 'p_recency_b_1yr',
       'p_recency_c_1yr+'],
      dtype='object')
```

6. Splitting Training and Testing Data:

In [445...
```
from sklearn.model_selection import train_test_split
```

In [446...
```
# Splitting Train and Test data:

df_train , df_test = train_test_split(df, train_size=0.8, random_state=100, stratify=df
```

In [447...
```
print(df_train.shape)
print(df_test.shape)
```

```
(8929, 41)
(2233, 41)
```

In [448...
```
print(df_train["deposit"].value_counts(normalize=True))
print("-"*20)
print(df_test["deposit"].value_counts(normalize=True))
```

```
0    0.526151
1    0.473849
Name: deposit, dtype: float64
--------------------
0    0.526198
1    0.473802
Name: deposit, dtype: float64
```

7. Feature Scaling:

In [449...
```
from sklearn.preprocessing import MinMaxScaler
```

In [450...
```
scaler = MinMaxScaler()
```

In [451...
```
df_train[["age","balance"]].describe()
```

Out[451...

	age	balance

	age	balance
count	8929.000000	8929.000000
mean	41.147945	1509.395117
std	11.848750	3193.242289
min	18.000000	-6847.000000
25%	32.000000	116.000000
50%	39.000000	541.000000
75%	49.000000	1693.000000
max	95.000000	81204.000000

In [452…

```
df_train[["age","balance"]] = scaler.fit_transform(df_train[["age","balance"]])
df_train.head()
```

Out[452…

	age	default	balance	housing	loan	deposit	job_blue-collar	job_entrepreneur	job_housemaic
7976	0.376623	0	0.101930	1	0	0	1	0	(
10101	0.324675	0	0.117716	0	0	1	0	0	(
9070	0.194805	0	0.155478	0	0	1	0	0	(
3546	0.376623	0	0.106154	0	0	0	0	0	(
465	0.142857	0	0.085644	1	1	1	1	0	(

In [453…

```
df_train.describe()
```

Out[453…

	age	default	balance	housing	loan	deposit	job_blue-collar	job_
count	8929.000000	8929.000000	8929.000000	8929.000000	8929.000000	8929.000000	8929.000000	
mean	0.300623	0.016351	0.094904	0.477097	0.130250	0.473849	0.177959	
std	0.153880	0.126829	0.036266	0.499503	0.336597	0.499344	0.382500	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.181818	0.000000	0.079079	0.000000	0.000000	0.000000	0.000000	
50%	0.272727	0.000000	0.083906	0.000000	0.000000	0.000000	0.000000	
75%	0.402597	0.000000	0.096989	1.000000	0.000000	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

In [454…

```
# for test data:
```

```
df_test[["age","balance"]] = scaler.transform(df_test[["age","balance"]])
```

In [455…
```
df_test.describe()
```

Out[455…

	age	default	balance	housing	loan	deposit	job_blue-collar	job_
count	2233.000000	2233.000000	2233.000000	2233.000000	2233.000000	2233.000000	2233.000000	
mean	0.306076	0.009852	0.095991	0.457232	0.133005	0.473802	0.158979	
std	0.157990	0.098790	0.038053	0.498279	0.339656	0.499425	0.365738	
min	0.000000	0.000000	0.051845	0.000000	0.000000	0.000000	0.000000	
25%	0.181818	0.000000	0.079306	0.000000	0.000000	0.000000	0.000000	
50%	0.272727	0.000000	0.084428	0.000000	0.000000	0.000000	0.000000	
75%	0.402597	0.000000	0.097989	1.000000	0.000000	1.000000	0.000000	
max	0.974026	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8. Model Building:

In [456…
```
x_train = df_train.drop("deposit", axis=1)
y_train = df_train["deposit"]
x_test = df_test.drop("deposit", axis=1)
y_test = df_test["deposit"]
```

In [457…
```
print(x_train.shape)
print("-"*20)
print(x_test.shape)
```

```
(8929, 40)
--------------------
(2233, 40)
```

- Model Building - Logistic Regression:

In [458…
```
from sklearn.linear_model import LogisticRegression
```

In [459…
```
logreg = LogisticRegression(random_state=100)
```

In [460…
```
logreg.fit(x_train,y_train)
```

Out[460…
```
LogisticRegression(random_state=100)
```

- Model Evaluation on Train set:

In [461…
```
y_train_pred = logreg.predict(x_train)
```

In [462…
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [463…
```
print(accuracy_score(y_train,y_train_pred))
```

```
0.7034382349647217
```

In [464…
```
print(confusion_matrix(y_train,y_train_pred))
```

```
[[3837  861]
 [1787 2444]]
```

In [465…
```
print(classification_report(y_train,y_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.68      0.82      0.74      4698
           1       0.74      0.58      0.65      4231

    accuracy                           0.70      8929
   macro avg       0.71      0.70      0.70      8929
weighted avg       0.71      0.70      0.70      8929
```

- Model Evaluation on Test Set:

In [466…
```
y_test_pred = logreg.predict(x_test)
```

In [467…
```
print(accuracy_score(y_test,y_test_pred))
```

```
0.7093596059113301
```

- Model Building - Random Forest

In [468…
```
from sklearn.ensemble import RandomForestClassifier
```

In [469…
```
rf = RandomForestClassifier(random_state=100,n_estimators = 50, oob_score = True)
```

In [470…
```
rf.fit(x_train,y_train)
```

Out[470…
```
RandomForestClassifier(n_estimators=50, oob_score=True, random_state=100)
```

- Model Evaluation on Train set:

In [471…
```
y_train_pred = rf.predict(x_train)
```

In [472…
```
print(accuracy_score(y_train, y_train_pred))
```
```
0.9985440698846455
```

- Model Evaluation on Test Set:

In [473…
```
y_test_pred = rf.predict(x_test)
```

In [474…
```
print(accuracy_score(y_test,y_test_pred))
```
```
0.7008508732646663
```

9. Cross Validation:

In [475…
```
from sklearn.model_selection import cross_val_score
```

In [476…
```
# Crosss Validation - Logistic Regression (Accuracy metric):

cross_val_score(logreg,x_train, y_train, cv = 5, n_jobs=-1)
```
Out[476…
```
array([0.6825308 , 0.69988802, 0.6987682 , 0.70828667, 0.70140056])
```

In [477…
```
cross_val_score(logreg,x_train, y_train, cv = 5, n_jobs=-1).mean()
```
Out[477…
```
0.698174848886923
```

In [478…
```
# Cross Validation - Random Forest(Accuracy metric):
cross_val_score(rf,x_train, y_train, cv = 5, n_jobs=-1)
```
Out[478…
```
array([0.68868981, 0.6881299 , 0.69708847, 0.693729  , 0.69467787])
```

In [479…
```
cross_val_score(rf,x_train, y_train, cv = 5, n_jobs=-1).mean()
```
Out[479…
```
0.6924630098399943
```

NOTE:Logistic Regression Algorithm is slightly doing better compared to the Random Forest Algorithm

- OOB Score in Random Forest Model:

In [480…
```
rf.oob_score_     # >>> OOB in RF is somewhat similar to cross Validation score
```
Out[480…
```
0.6820472617314369
```

In [481…
```
import sklearn
```

In [482…
```
# Choosing Other metrics to check cross validation score:
sklearn.metrics.SCORERS.keys()
```

Out[482…
```
dict_keys(['explained_variance', 'r2', 'max_error', 'neg_median_absolute_error', 'neg_mean_absolute_error', 'neg_mean_absolute_percentage_error', 'neg_mean_squared_error', 'neg_mean_squared_log_error', 'neg_root_mean_squared_error', 'neg_mean_poisson_deviance', 'neg_mean_gamma_deviance', 'accuracy', 'top_k_accuracy', 'roc_auc', 'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted', 'roc_auc_ovo_weighted', 'balanced_accuracy', 'average_precision', 'neg_log_loss', 'neg_brier_score', 'adjusted_rand_score', 'rand_score', 'homogeneity_score', 'completeness_score', 'v_measure_score', 'mutual_info_score', 'adjusted_mutual_info_score', 'normalized_mutual_info_score', 'fowlkes_mallows_score', 'precision', 'precision_macro', 'precision_micro', 'precision_samples', 'precision_weighted', 'recall', 'recall_macro', 'recall_micro', 'recall_samples', 'recall_weighted', 'f1', 'f1_macro', 'f1_micro', 'f1_samples', 'f1_weighted', 'jaccard', 'jaccard_macro', 'jaccard_micro', 'jaccard_samples', 'jaccard_weighted'])
```

In [483…
```
# Cross Validation - Random Forest (recall metric)
cross_val_score(rf,x_train,y_train, cv = 5, n_jobs=-1,scoring="recall")
```

Out[483…
```
array([0.61465721, 0.62529551, 0.63829787, 0.6068477 , 0.64184397])
```

10. Feature Selection:

- Recursive Feature Selection:

In [484…
```
from sklearn.feature_selection import RFE
```

In [485…
```
logreg = LogisticRegression(random_state=100)
```

In [486…
```
rfe = RFE(estimator=logreg, n_features_to_select=10)
```

In [487…
```
rfe = rfe.fit(x_train,y_train)
```

In [488…
```
rfe
```

Out[488…
```
RFE(estimator=LogisticRegression(random_state=100), n_features_to_select=10)
```

In [489…
```
rfe.support_
```

Out[489…
```
array([False, False,  True,  True, False, False, False, False, False,
       False, False, False,  True, False, False, False, False, False,
       False, False, False, False,  True, False,  True, False, False,
       False, False,  True, False, False,  True,  True, False,  True,
       False, False, False,  True])
```

In [490…
```
rfe.ranking_
```

```
array([17, 26,  1,  1, 11, 30, 21, 12, 22,  9, 20, 27,  1, 31, 15, 25, 10,
```

Out[490…
```
       24, 18, 16, 19,  8,  1,  3,  1,  7,  2,  4, 29,  1,  6,  5,  1,  1,
       13,  1, 28, 23, 14,  1])
```

In [491…
```
x_train.columns[rfe.support_]
```

Out[491…
```
Index(['balance', 'housing', 'job_student', 'contact_unknown', 'month_dec',
       'month_mar', 'month_oct', 'month_sep', 'poutcome_success',
       'p_recency_c_1yr+'],
      dtype='object')
```

In [492…
```
featuress = x_train.columns[rfe.support_]
featuress
```

Out[492…
```
Index(['balance', 'housing', 'job_student', 'contact_unknown', 'month_dec',
       'month_mar', 'month_oct', 'month_sep', 'poutcome_success',
       'p_recency_c_1yr+'],
      dtype='object')
```

In [495…
```
x_train2 = x_train[featuress]
```

- Evaluation - Cross valuation Score

In [496…
```
# Cross Valuation Score for Logistic Regression:

cross_val_score(logreg,x_train2, y_train, n_jobs=-1)
```

Out[496…
```
array([0.67805151, 0.68421053, 0.68085106, 0.66797312, 0.69915966])
```

Cross Validation for Feature Selection:

In [497…
```
num_features = x_train.shape
num_features[1]
```

Out[497…
```
40
```

In [498…
```
cv_scores = []
```

In [499…
```
logreg = LogisticRegression(random_state=100)
```

In [500…
```
for features in range(1,num_features[1]+1):
    rfe = RFE(logreg, n_features_to_select=features)
    scores = cross_val_score(rfe,x_train,y_train, cv=4)
    cv_scores.append(scores.mean())
```

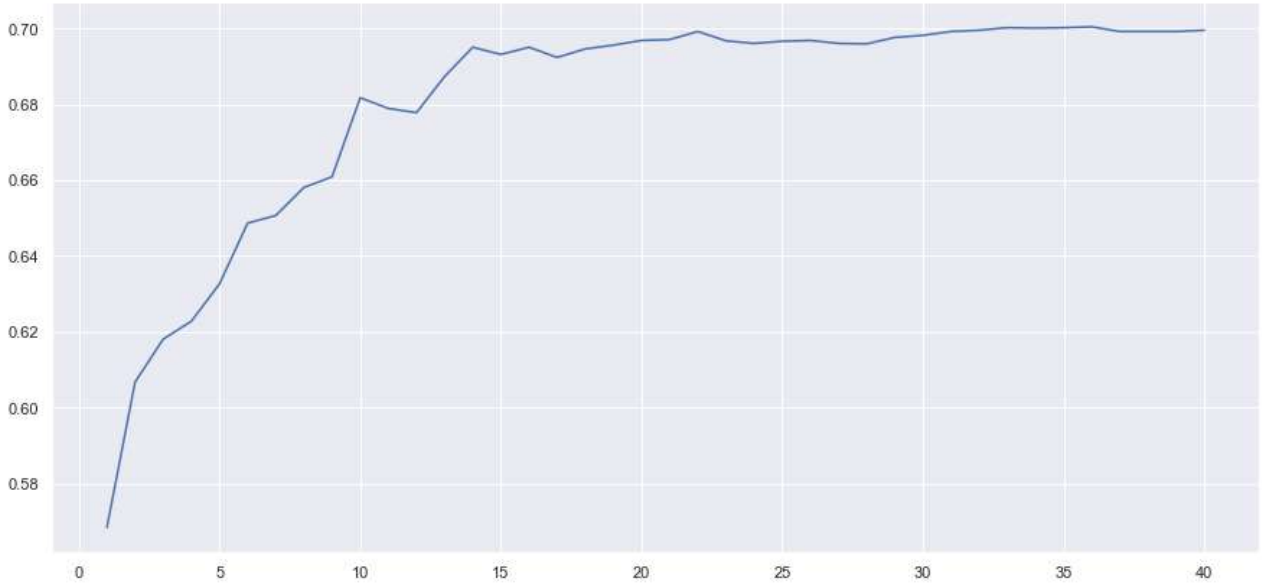
In [501…
```
print(cv_scores)
```

```
[0.568488445956466, 0.6067873234169119, 0.6180988937523977, 0.6228023922684657, 0.632658
1703736877, 0.6486731890652915, 0.6506890672977993, 0.6580806375369779, 0.66088036530889
69, 0.6817115919243283, 0.6789109111133582, 0.6777919931076215, 0.6873109973082164, 0.69
```

```
50389903323719, 0.6931347681486725, 0.6950388398525218, 0.692351721168462, 0.69459171405
77874, 0.6955995779341163, 0.6968318574269631, 0.6970558717638807, 0.6991837571648472,
0.6967203016980548, 0.6960483088472522, 0.6966081942096958, 0.6968322587065634, 0.696048
007887552, 0.6959357499193427, 0.6976157571263245, 0.6981758431285684, 0.69918345620514
7, 0.6995196281903735, 0.700191621041176, 0.7000797643525675, 0.7001918216809763, 0.7004
158360178938, 0.6991837571648472, 0.6991837070048972, 0.6991837571648473, 0.699519778670
2236]
```

In [502…]
```
plt.figure(figsize=(15,7))
plt.plot(range(1,num_features[1]+1), cv_scores)
```

Out[502…]
```
[<matplotlib.lines.Line2D at 0x13e759b5430>]
```



Using RFECV:

In [503…]
```
from sklearn.feature_selection import RFECV
```

In [504…]
```
rfecv = RFECV(estimator=logreg, cv = 4)
```

In [505…]
```
%%time
rfecv.fit(x_train,y_train)
```

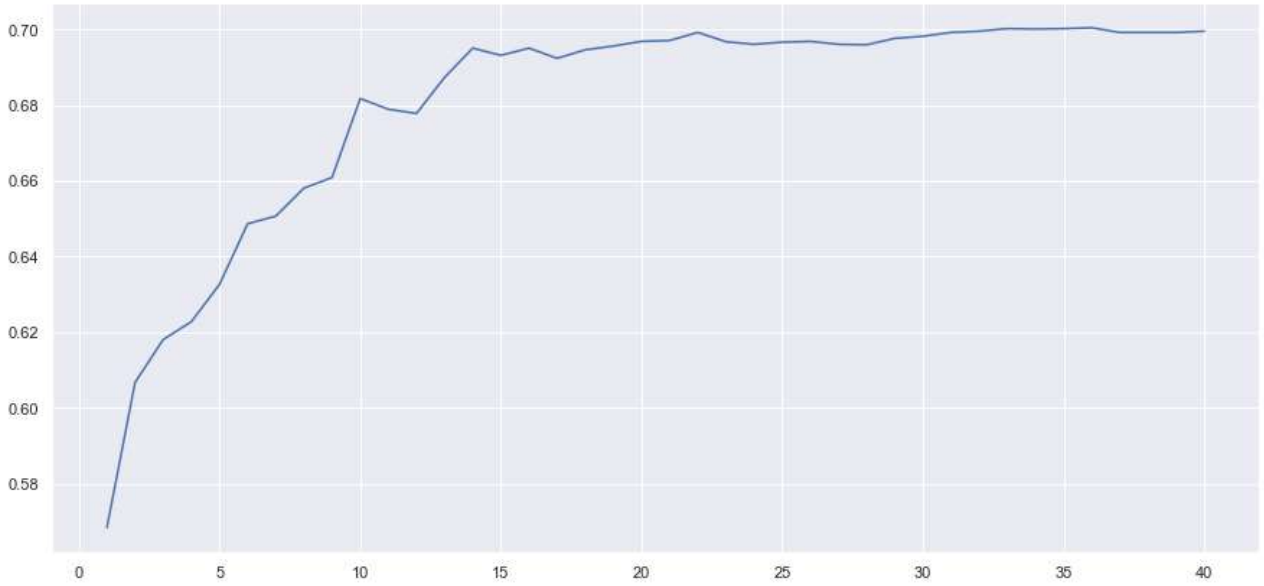
Out[505…]
```
Wall time: 9.6 s
RFECV(cv=4, estimator=LogisticRegression(random_state=100))
```

In [506…]
```
rfecv.grid_scores_
```

Out[506…]
```
array([0.56848845, 0.60678732, 0.61809889, 0.62280239, 0.63265817,
       0.64867319, 0.65068907, 0.65808064, 0.66088037, 0.68171159,
       0.67891091, 0.67779199, 0.687311  , 0.69503899, 0.69313477,
       0.69503884, 0.69235172, 0.69459171, 0.69559958, 0.69683186,
       0.69705587, 0.69918376, 0.6967203 , 0.69604831, 0.69660819,
       0.69683226, 0.69604801, 0.69593575, 0.69761576, 0.69817584,
       0.69918346, 0.69951963, 0.70019162, 0.70007976, 0.70019182,
       0.70041584, 0.69918376, 0.69918371, 0.69918376, 0.69951978])
```

In [507…
```
plt.figure(figsize=(15,7))
plt.plot(range(1,num_features[1]+1), rfecv.grid_scores_)
```

Out[507…
```
[<matplotlib.lines.Line2D at 0x13e75f16ca0>]
```



In [508…
```
rfecv.n_features_
```

Out[508…
```
36
```

11. Hyper Tuning - Cross Validation:

In [509…
```
from sklearn.ensemble import RandomForestClassifier
```

In [510…
```
rf = RandomForestClassifier(random_state=100, n_jobs=-1)
```

In [511…
```
from sklearn.model_selection import GridSearchCV
```

In [512…
```
params = {"max_depth": [3, 5, 10, 15, 20],
          "max_features": [3,5,7,11,15],
          "min_samples_leaf":[20,50,100,200,400],
          "n_estimators": [10,25,50,80,100]}
```

In [513…
```
Grid_search = GridSearchCV(estimator=rf, param_grid=params, n_jobs=-1, verbose=1,cv = 5
```

In [514…
```
Grid_search.fit(x_train,y_train)
```

Out[514…
```
Fitting 5 folds for each of 625 candidates, totalling 3125 fits
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(n_jobs=-1, random_state=100),
             n_jobs=-1,
```

```
                param_grid={'max_depth': [3, 5, 10, 15, 20],
                            'max_features': [3, 5, 7, 11, 15],
                            'min_samples_leaf': [20, 50, 100, 200, 400],
                            'n_estimators': [10, 25, 50, 80, 100]},
                return_train_score=True, verbose=1)
```

In [515…
```
Grid_search.best_score_
```

Out[515…
```
0.7126220432181831
```

In [516…
```
Grid_search.best_estimator_
```

Out[516…
```
RandomForestClassifier(max_depth=20, max_features=5, min_samples_leaf=20,
                       n_jobs=-1, random_state=100)
```

In [517…
```
cv_df = pd.DataFrame(Grid_search.cv_results_)
cv_df
```

Out[517…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_featur
0	0.145459	0.007614	0.047424	0.013688	3	
1	0.253358	0.033343	0.144128	0.041028	3	
2	0.478714	0.051285	0.104104	0.023726	3	
3	0.831894	0.082943	0.159304	0.055154	3	
4	1.193332	0.051706	0.160804	0.030328	3	
…	…	…	…	…	…	
620	0.376438	0.190787	0.241471	0.188566	20	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_featur
621	0.391153	0.053283	0.169566	0.098149	20	
622	0.627933	0.074006	0.171864	0.072934	20	
623	1.010517	0.108977	0.424112	0.184221	20	
624	1.209111	0.143366	0.124860	0.040749	20	

625 rows × 24 columns

In [518…

```
cv_df = cv_df.sort_values(by = ["rank_test_score"])
cv_df.head()
```

Out[518…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_featur
529	1.224921	0.074319	0.222548	0.071012	20	
527	0.738486	0.111438	0.242092	0.102166	20	
402	0.429065	0.050311	0.121760	0.038481	15	
404	0.968691	0.084603	0.133920	0.024499	15	
528	0.961480	0.161496	0.393377	0.106302	20	

In [519…

```
sel_cols = ["param_max_depth","param_max_features","param_min_samples_leaf","param_n_es
```

```
"rank_test_score", "mean_train_score","mean_test_score"]
```

In [520…

```
cv_df.sort_values(by = "rank_test_score")[sel_cols].head(20)
```

Out[520…

	param_max_depth	param_max_features	param_min_samples_leaf	param_n_estimators	rank_test_sco
529	20	5	20	100	
527	20	5	20	50	
402	15	5	20	50	
404	15	5	20	100	
528	20	5	20	80	
403	15	5	20	80	
278	10	5	20	80	
376	15	3	20	25	
428	15	7	20	80	
279	10	5	20	100	
377	15	3	20	50	
429	15	7	20	100	
378	15	3	20	80	
526	20	5	20	25	
579	20	11	20	100	
400	15	5	20	10	
401	15	5	20	25	
504	20	3	20	100	
277	10	5	20	50	
503	20	3	20	80	

In [521…

```
# Understanding the better: Effect of Hyper Parameter:

cv_df.columns
```

Out[521…

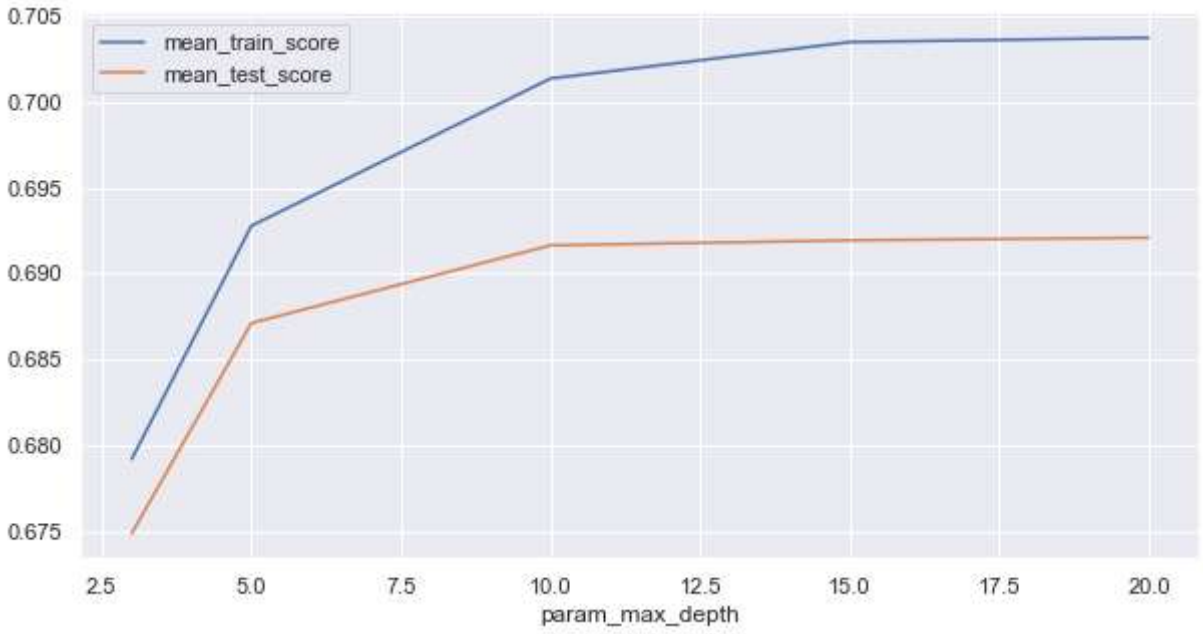
```
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_max_depth', 'param_max_features', 'param_min_samples_leaf',
       'param_n_estimators', 'params', 'split0_test_score',
       'split1_test_score', 'split2_test_score', 'split3_test_score',
       'split4_test_score', 'mean_test_score', 'std_test_score',
       'rank_test_score', 'split0_train_score', 'split1_train_score',
       'split2_train_score', 'split3_train_score', 'split4_train_score',
       'mean_train_score', 'std_train_score'],
      dtype='object')
```

In [522…
```
cv_df.groupby("param_max_depth")["mean_train_score", "mean_test_score"].mean()
```

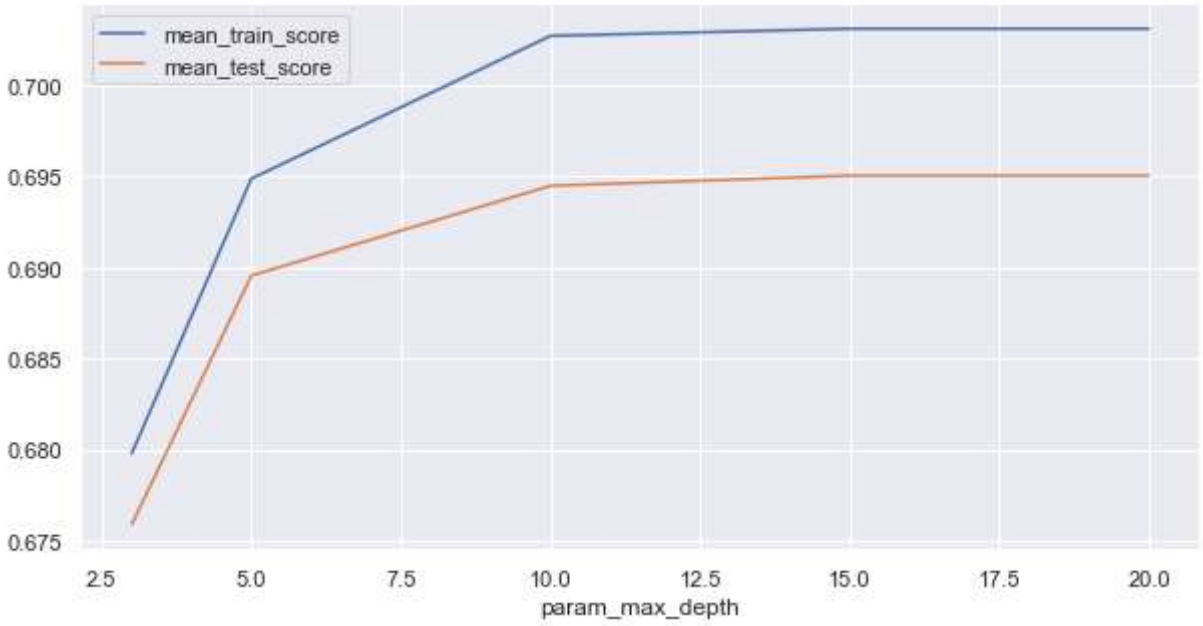
Out[522…

param_max_depth	mean_train_score	mean_test_score
3	0.679182	0.674851
5	0.692787	0.687117
10	0.701377	0.691649
15	0.703504	0.691942
20	0.703756	0.692096

In [523…
```
cv_df.groupby("param_max_depth")["mean_train_score", "mean_test_score"].mean().plot(fig
plt.show()
```

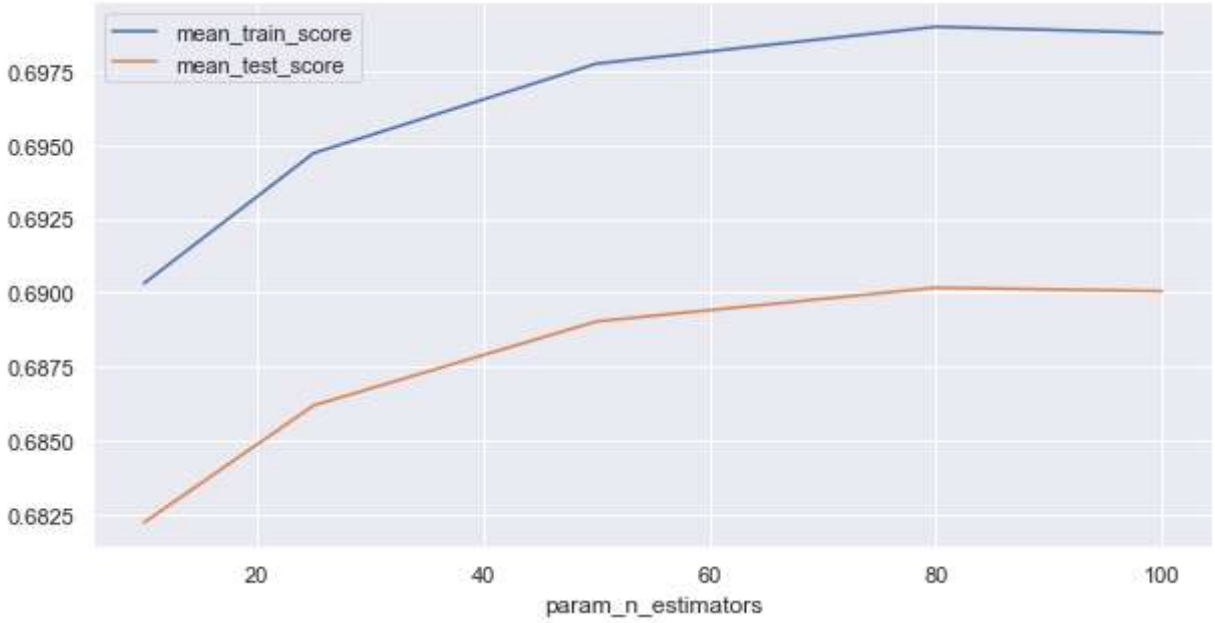


In [524…
```
cv_df.groupby("param_max_depth")["mean_train_score", "mean_test_score"].agg(np.median).
plt.show()
```



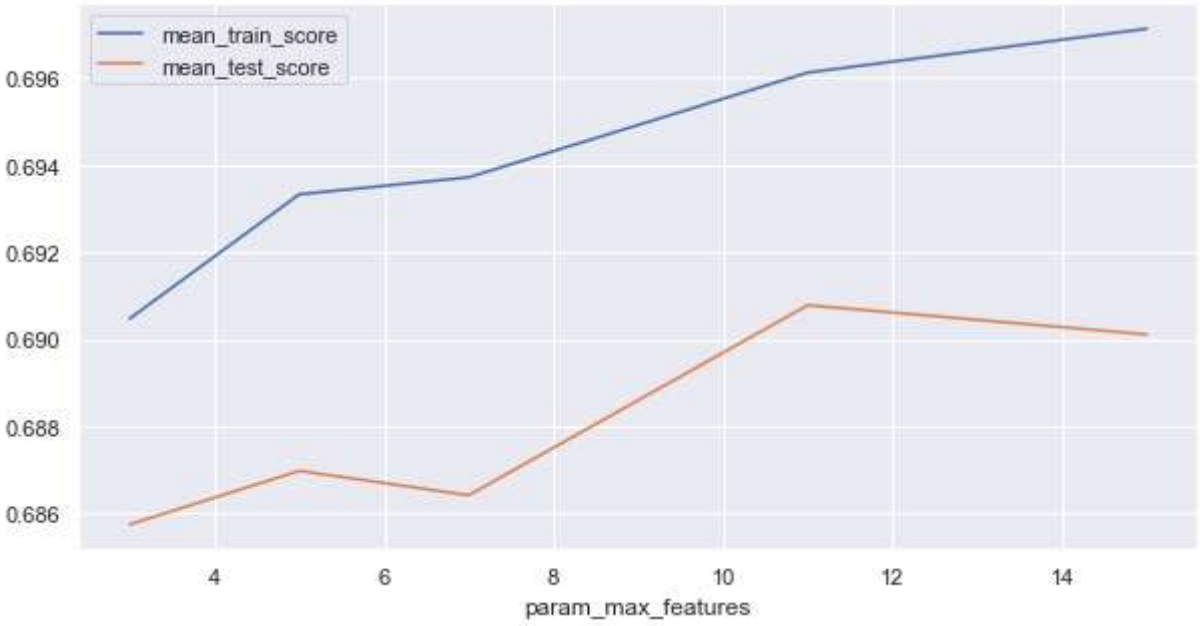
In [525…

```
cv_df.groupby("param_n_estimators")["mean_train_score", "mean_test_score"].agg(np.mean)
plt.show()
```



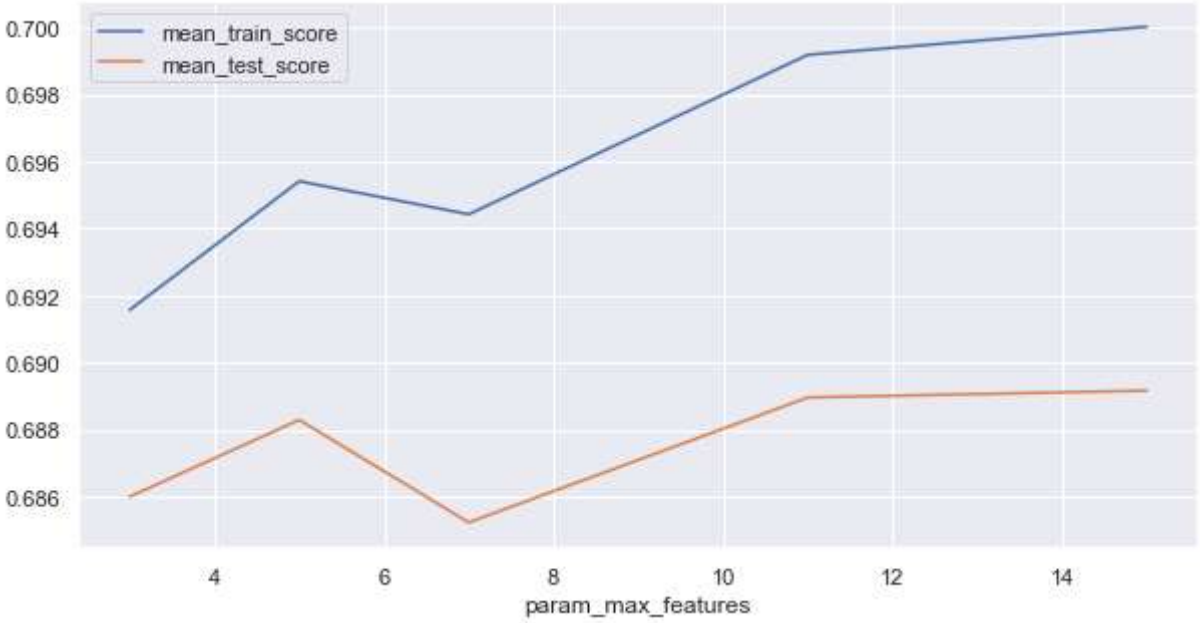
In [526…

```
cv_df.groupby("param_max_features")["mean_train_score", "mean_test_score"].agg(np.media
plt.show()
```



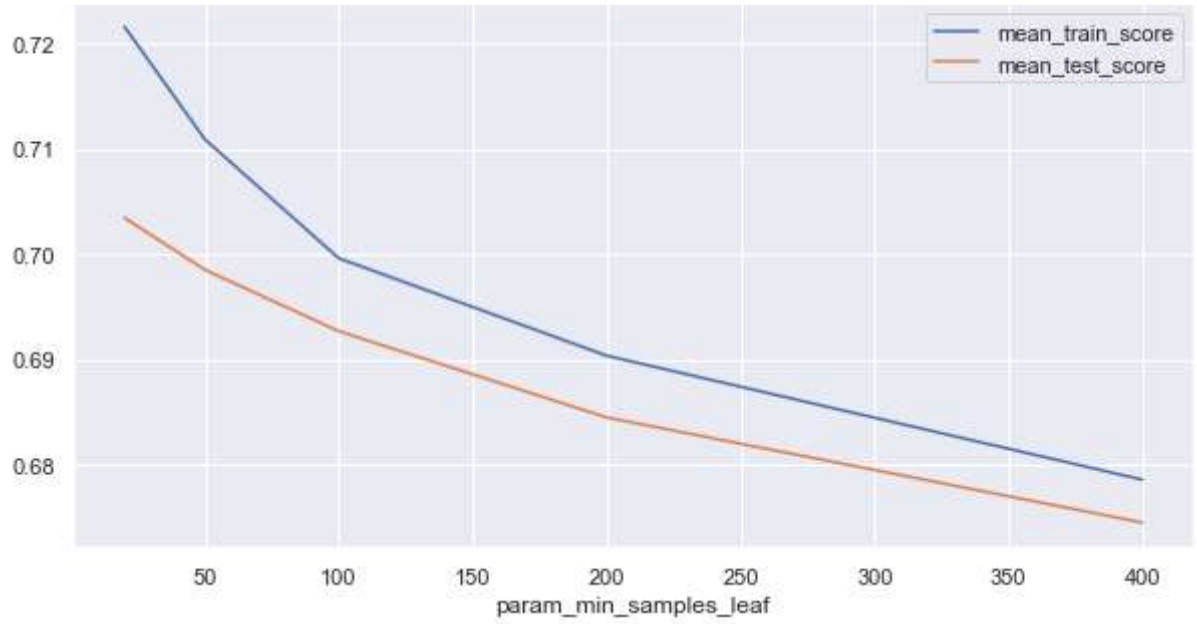
In [527…

```
cv_df.groupby("param_max_features")["mean_train_score", "mean_test_score"].agg(np.mean)
plt.show()
```



In [528…

```
cv_df.groupby("param_min_samples_leaf")["mean_train_score", "mean_test_score"].agg(np.m
plt.show()
```



- Fine Tuning using Grid Search:

In [529…
```
params  = {"min_samples_leaf" : [5,10,20,50],
           "n_estimators": [50,60,70],
           "max_features": [10,12,14,16]}
```

In [530…
```
rf = RandomForestClassifier(max_depth=12, random_state=100, n_jobs=-1)
```

In [531…
```
Grid_search2 = GridSearchCV(estimator=rf,
                            param_grid=params,
                            verbose=1,
                            cv = 5,
                            return_train_score=True,
                            n_jobs=-1)
```

In [532…
```
Grid_search2.fit(x_train,y_train)
```

Out[532…
```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(max_depth=12, n_jobs=-1,
                                              random_state=100),
             n_jobs=-1,
             param_grid={'max_features': [10, 12, 14, 16],
                         'min_samples_leaf': [5, 10, 20, 50],
                         'n_estimators': [50, 60, 70]},
             return_train_score=True, verbose=1)
```

In [533…
```
Grid_search2.best_score_
```

Out[533…
```
0.7129580522018438
```

In [534…
```
Grid_search2.best_estimator_
```

Out[534…
```
RandomForestClassifier(max_depth=12, max_features=12, min_samples_leaf=5,
                       n_estimators=70, n_jobs=-1, random_state=100)
```

- Hyper Tuning - RandomizedSearchCV:

In [540…
```
from sklearn.model_selection import RandomizedSearchCV
```

In [541…
```
params = {"max_depth":range(3,20),
          "max_features":range(3,17),
          "min_samples_leaf":range(20,400,50),
          "n_estimators":range(10,101,10)}
```

In [542…
```
rscv = RandomizedSearchCV(estimator=rf, param_distributions=params,
                          verbose=1, cv=5,
                          return_train_score=True, n_jobs=-1,
                          n_iter=50)
```

In [543…
```
rscv.fit(x_train,y_train)
```

Out[543…
```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
RandomizedSearchCV(cv=5,
                   estimator=RandomForestClassifier(max_depth=12, n_jobs=-1,
                                                    random_state=100),
                   n_iter=50, n_jobs=-1,
                   param_distributions={'max_depth': range(3, 20),
                                        'max_features': range(3, 17),
                                        'min_samples_leaf': range(20, 400, 50)},
                   return_train_score=True, verbose=1)
```

In [544…
```
rscv.best_score_
```

Out[544…
```
0.7091493439481056
```

In [545…
```
Grid_search.best_score_
```

Out[545…
```
0.7126220432181831
```

In [546…
```
Grid_search2.best_score_
```

Out[546…
```
0.7129580522018438
```

12. Exrtacting the Best Model:

In [547…
```
rf_best = Grid_search2.best_estimator_
rf_best
```

Out[547…
```
RandomForestClassifier(max_depth=12, max_features=12, min_samples_leaf=5,
                       n_estimators=70, n_jobs=-1, random_state=100)
```

In [548…
```
y_test_pred = rf_best.predict(x_test)
```

In [549…
```
y_test_pred
```

Out[549…
```
array([0, 1, 0, ..., 1, 1, 0], dtype=int64)
```

In [550…
```
accuracy_score(y_test, y_test_pred)
```

Out[550…
```
0.7299596954769368
```

In [551…
```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.71      0.82      0.76      1175
           1       0.76      0.63      0.69      1058

    accuracy                           0.73      2233
   macro avg       0.74      0.72      0.72      2233
weighted avg       0.73      0.73      0.73      2233
```

Final Model is having accuracy of 72%