Import data analyis Libraries

In [1]:
```
import time
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Import spliting libraries from scikit_learn

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

import selection libraries

In [3]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [4]:
```
from imblearn.over_sampling import SMOTE
```

import pipeline libraries

In [5]:
```
from imblearn.pipeline import Pipeline as imbpipeline
from sklearn.pipeline import Pipeline
```

In [6]:
```
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
```

import feature_engineer libraries

In [7]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

import Metrics libraries

In [8]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
```

Import scikit-learn libraries

In [9]:
```
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LinearRegression
```

remove warnings

In [10]:
```
import warnings
warnings.filterwarnings("ignore")
```

Load dataset

In [11]:
```
df = pd.read_csv("online_shoppers_intention.csv")
```

Access first five rows

In [12]:
```
df.head()
```

Out[12]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	Prod
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

Access last five rows

In [13]:
```
df.tail()
```

Out[13]:

	Administrative	Administrative_Duration	Informational	Informational_Duration
12325	3	145.0	0	0.0
12326	0	0.0	0	0.0
12327	0	0.0	0	0.0
12328	4	75.0	0	0.0
12329	0	0.0	0	0.0

identify no.of columns and rows

In [14]:
```
df.shape
```
Out[14]:
```
(12330, 18)
```

rows

In [15]:
```
df.shape[0]
```
Out[15]:
```
12330
```

columns

In [125…
```
df.shape[1]
```
Out[125…
```
18
```

In [126…
```
df.columns
```
Out[126…
```
Index(['Administrative', 'Administrative_Duration', 'Informational',
       'Informational_Duration', 'ProductRelated', 'ProductRelated_Duration',
       'BounceRates', 'ExitRates', 'PageValues', 'SpecialDay', 'Month',
       'OperatingSystems', 'Browser', 'Region', 'TrafficType', 'Weekend',
       'Revenue', 'Returning_Visitor'],
      dtype='object')
```

In [127…
```
for column in df.columns:
    print(column)
```

```
Administrative
Administrative_Duration
Informational
Informational_Duration
ProductRelated
ProductRelated_Duration
BounceRates
ExitRates
PageValues
SpecialDay
Month
OperatingSystems
Browser
Region
TrafficType
Weekend
Revenue
Returning_Visitor
```

desribing about dataset

In [17]:
```
df.describe().round(2)
```

Out[17]:

	Administrative	Administrative_Duration	Informational	Informational_Duration
count	12330.00	12330.00	12330.00	12330.00
mean	2.32	80.82	0.50	34.47
std	3.32	176.78	1.27	140.75
min	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00
50%	1.00	7.50	0.00	0.00
75%	4.00	93.26	0.00	0.00
max	27.00	3398.75	24.00	2549.38

dataset information

In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object 
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  bool   
 17  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

number of unique values columns wise

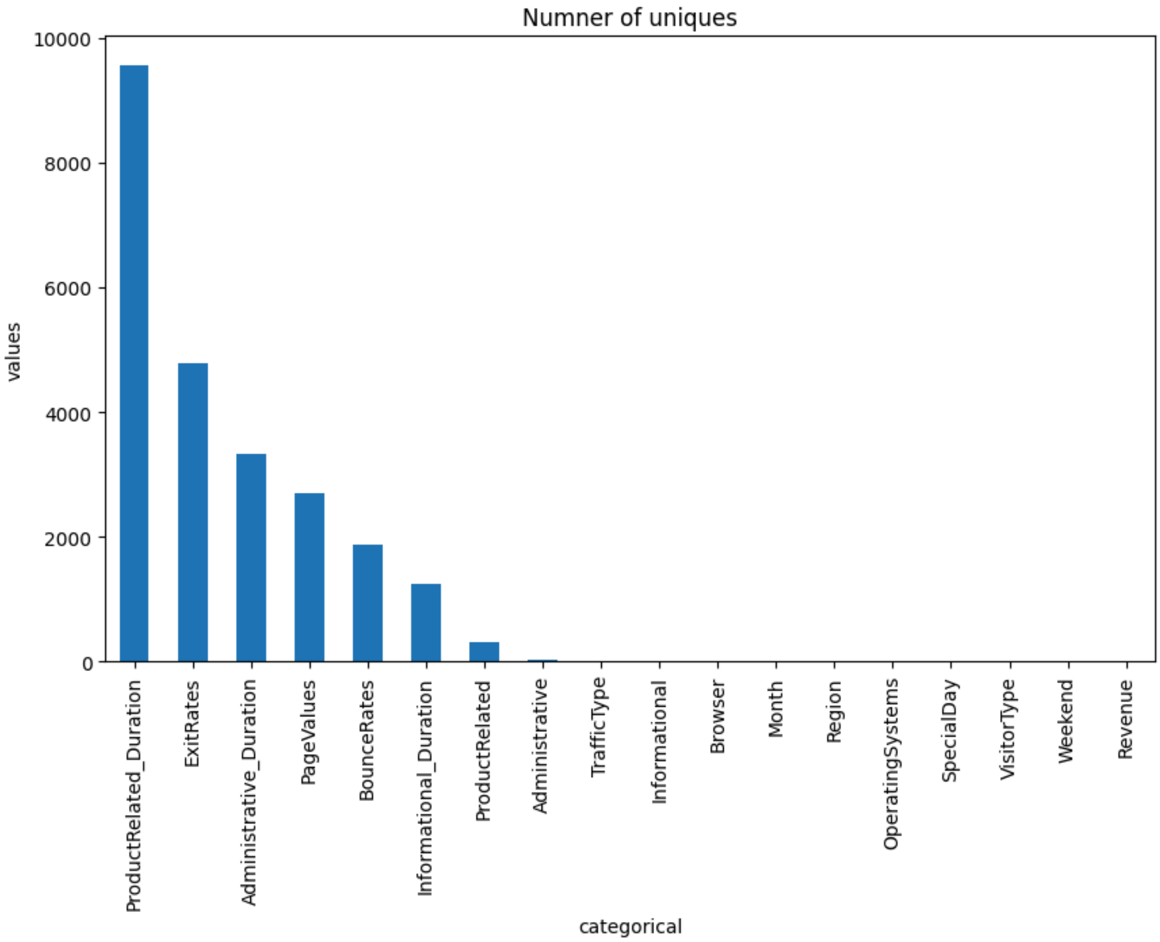
In [19]:
```
df.nunique().sort_values(ascending=False)
```

Out[19]:
```
ProductRelated_Duration    9551
ExitRates                  4777
Administrative_Duration    3335
PageValues                 2704
BounceRates                1872
Informational_Duration     1258
ProductRelated              311
Administrative               27
TrafficType                  20
Informational                17
Browser                      13
Month                        10
Region                        9
OperatingSystems              8
SpecialDay                    6
VisitorType                   3
Weekend                       2
Revenue                       2
dtype: int64
```

Visualazation no.of uniques

In [20]:
```
df.nunique().sort_values(ascending=False).plot.bar(figsize=(10,6))
plt.title("Numner of uniques")
plt.xlabel("categorical")
plt.ylabel("values")
```

Out[20]: Text(0, 0.5, 'values')



types of columns

In [21]:
```
df.dtypes
```

Out[21]:
```
Administrative               int64
Administrative_Duration    float64
Informational                int64
Informational_Duration     float64
ProductRelated               int64
ProductRelated_Duration    float64
BounceRates                float64
ExitRates                  float64
PageValues                 float64
SpecialDay                 float64
Month                       object
OperatingSystems             int64
Browser                      int64
Region                       int64
TrafficType                  int64
VisitorType                 object
Weekend                       bool
Revenue                       bool
dtype: object
```

Access Month column

In [22]:
```
df.Month.unique()
```
Out[22]:
```
array(['Feb', 'Mar', 'May', 'Oct', 'June', 'Jul', 'Aug', 'Nov', 'Sep',
       'Dec'], dtype=object)
```

Applying feature_eng techniques

In [24]:
```
Ordinal_encoder = OrdinalEncoder()
```
In [25]:
```
df["Month"]=Ordinal_encoder.fit_transform(df[["Month"]])
```
In [26]:
```
df.Month
```
Out[26]:
```
0        2.0
1        2.0
2        2.0
3        2.0
4        2.0
        ...
12325    1.0
12326    7.0
12327    7.0
12328    7.0
12329    7.0
Name: Month, Length: 12330, dtype: float64
```

Access Weekend and Revenue columns

In [27]:
```
df.Weekend.dtype
```
Out[27]:
```
dtype('bool')
```
In [28]:
```
df.Weekend.unique()
```
Out[28]:
```
array([False,  True])
```
In [29]:
```
df.Revenue.dtype
```
Out[29]:
```
dtype('bool')
```
In [30]:
```
df.Revenue.unique()
```
Out[30]:
```
array([False,  True])
```

Convert Bool values into integer values

In [31]:
```
df["Weekend"] = df.Weekend.replace((True,False),(1,0))
df["Revenue"] = df.Revenue.replace((True,False),(1,0))
```
In [32]:
```
df.iloc[0:6]
```

Out[32]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	Prod
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	
5	0	0.0	0	0.0	

In [128…

```
df.dtypes
```

Out[128…

```
Administrative               int64
Administrative_Duration    float64
Informational                int64
Informational_Duration     float64
ProductRelated               int64
ProductRelated_Duration    float64
BounceRates                float64
ExitRates                  float64
PageValues                 float64
SpecialDay                 float64
Month                      float64
OperatingSystems             int64
Browser                      int64
Region                       int64
TrafficType                  int64
Weekend                      int64
Revenue                      int64
Returning_Visitor            int64
dtype: object
```

TrafficType column

In [33]:

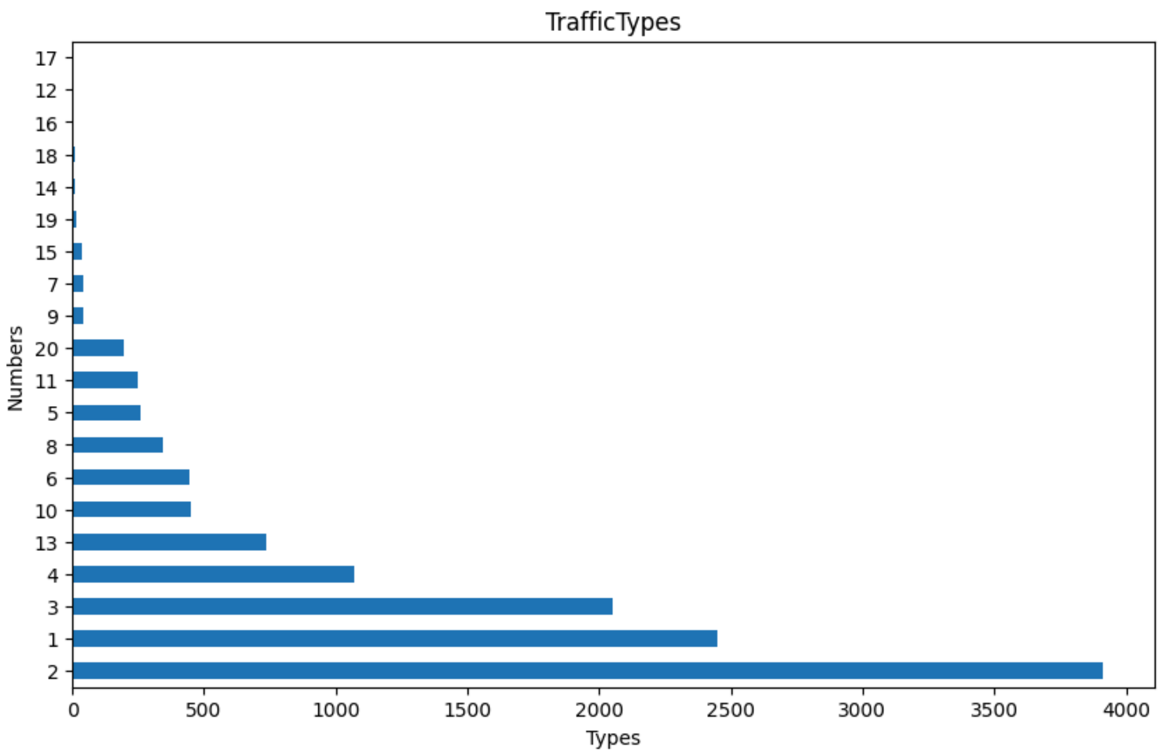
```
df.TrafficType.value_counts(ascending=False)
```

Out[33]:
```
TrafficType
2     3913
1     2451
3     2052
4     1069
13     738
10     450
6      444
8      343
5      260
11     247
20     198
9       42
7       40
15      38
19      17
14      13
18      10
16       3
12       1
17       1
Name: count, dtype: int64
```

Visualazation of Value counts

In [34]:
```python
df.TrafficType.value_counts().plot.barh(figsize=(10,6))
plt.title("TrafficTypes")
plt.xlabel("Types")
plt.ylabel("Numbers")
```

Out[34]:
```
Text(0, 0.5, 'Numbers')
```



Access Region column

In [35]:
```
df.Region.unique()
```

Out[35]:
```
array([1, 9, 2, 3, 4, 5, 6, 7, 8])
```

In [36]:
```
df.Region.value_counts()
```

Out[36]:
```
Region
1    4780
3    2403
4    1182
2    1136
6     805
7     761
9     511
8     434
5     318
Name: count, dtype: int64
```

In [37]:
```
df.Region.sum()
```

Out[37]:
```
np.int64(38807)
```

In [38]:
```
df.Region.min()
```

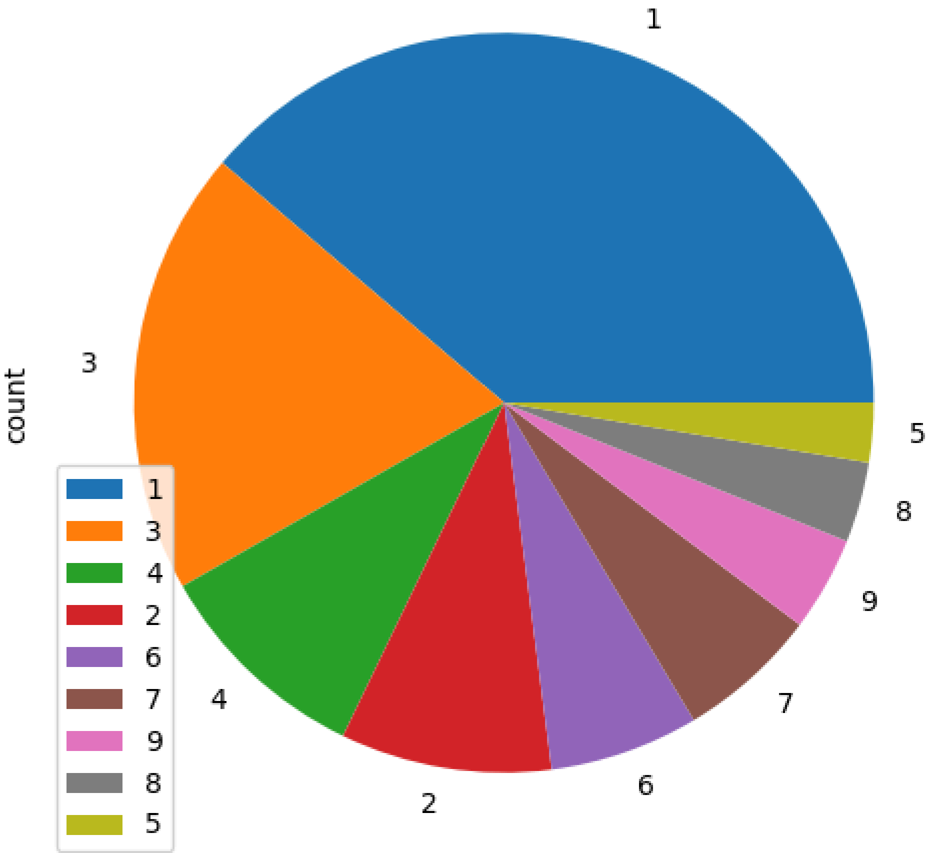
Out[38]:
```
np.int64(1)
```

In [39]:
```
df.Region.max()
```

Out[39]:
```
np.int64(9)
```

In [40]:
```
df.Region.value_counts().plot.pie(figsize=(10,6))
plt.legend()
```

Out[40]:
```
<matplotlib.legend.Legend at 0x1b97602f8c0>
```



In [41]:
```
df.VisitorType.unique()
```

Out[41]:
```
array(['Returning_Visitor', 'New_Visitor', 'Other'], dtype=object)
```

In [42]:
```
df.VisitorType.value_counts()
```

Out[42]:
```
VisitorType
Returning_Visitor    10551
New_Visitor           1694
Other                   85
Name: count, dtype: int64
```

Adding column

In [43]:
```
condition =df.VisitorType == "Returning_Visitor"
```

In [44]:
```
df[condition]
```

Out[44]:

	Administrative	Administrative_Duration	Informational	Informational_Duration
0	0	0.0	0	0.0
1	0	0.0	0	0.0
2	0	0.0	0	0.0
3	0	0.0	0	0.0
4	0	0.0	0	0.0
...	...	...	...	...
12324	0	0.0	1	0.0
12325	3	145.0	0	0.0
12326	0	0.0	0	0.0
12327	0	0.0	0	0.0
12328	4	75.0	0	0.0

10551 rows × 18 columns

In [45]:
```
df["Returning_Visitor"] = np.where(condition,1,0)
```

In [46]:
```
df.head()
```

Out[46]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	Prod
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

In [47]:
```
df.shape
```

Out[47]: (12330, 19)

dropping VisitorType column

In [48]:
```
df=df.drop("VisitorType",axis='columns')
```

In [49]:
```
df.shape
```

Out[49]: (12330, 18)

correlation

In [50]:
```
correlation = df[df.columns[1:]].corr()["Revenue"]
```

In [51]:
```
correlation.sort_values(ascending=False)
```

Out[51]:
```
Revenue                    1.000000
PageValues                 0.492569
ProductRelated             0.158538
ProductRelated_Duration    0.152373
Informational              0.095200
Administrative_Duration    0.093587
Month                      0.080150
Informational_Duration     0.070345
Weekend                    0.029295
Browser                    0.023984
TrafficType               -0.005113
Region                    -0.011595
OperatingSystems          -0.014668
SpecialDay                -0.082305
Returning_Visitor         -0.103843
BounceRates               -0.150673
ExitRates                 -0.207071
Name: Revenue, dtype: float64
```

In [52]:
```
x=df.drop("Revenue",axis='columns')
y=df.Revenue

x_train,x_test,y_train,y_test=train_test_split(
                                                x,
                                                y,
                                                test_size=0.33,
                                                random_state=50
                                               )


def model_pipeline(X, model):

    n_c = X.select_dtypes(exclude=['object']).columns.tolist()
    c_c = X.select_dtypes(include=['object']).columns.tolist()

    numeric_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='constant')),
    ('scaler', MinMaxScaler())])

    categorical_pipeline = Pipeline([
    ('encoder', OneHotEncoder(handle_unknown='ignore'))])

    preprocessor = ColumnTransformer([
    ('numeric', numeric_pipeline, n_c),
    ('categorical', categorical_pipeline, c_c)], remainder='passthrough')

    final_steps = [
    ('preprocessor', preprocessor),
    ('smote', SMOTE(random_state=1)),
    ('feature_selection', SelectKBest(score_func = chi2, k = 6)), ('model', mode

    return imbpipeline(steps = final_steps)
```

In [83]:
```
def select_model(x, y, pipeline=None):
    classifiers = {}
    c_d1 = {"DummyClassifier": DummyClassifier(strategy='most_frequent')}
    classifiers.update(c_d1)

    c_d2 = {"linearRegression":LinearRegression()}
    classifiers.update(c_d2)

    c_d3={"Xgboost":XGBClassifier()}
    classifiers.update(c_d3)

    c_d4 = {"RandomForestClassifier": RandomForestClassifier()}
    classifiers.update(c_d4)

    c_d5 = {"DecisionTreeClassifier": DecisionTreeClassifier()}
    classifiers.update(c_d5)

    c_d6={"svc":SVC()}
    classifiers.update(c_d6)

    c_d7={"lightgbm":LGBMClassifier()}
    classifiers.update(c_d7)

    c_d8={"kneighbors":KNeighborsClassifier()}
    classifiers.update(c_d8)
```

```
    c_d9 = {"KNeighborsClassifier": KNeighborsClassifier()}
    classifiers.update(c_d9)

    c_d10={"extratree":ExtraTreeClassifier()}
    classifiers.update(c_d10)

    c_d11={"extratrees":ExtraTreesClassifier()}
    classifiers.update(c_d11)

    c_d12={"redge":RidgeClassifier()}
    classifiers.update(c_d12)

    c_d13={"SGDC":SGDClassifier()}
    classifiers.update(c_d13)

    c_d14 = {"SVC": SVC()}
    classifiers.update(c_d14)

    c_d15={"BernoulliNB":BernoulliNB()}
    classifiers.update(c_d15)

    mlpc = { "MLPClassifier (paper)": MLPClassifier(hidden_layer_sizes=(27, 50),
     max_iter=300,
     activation='relu',
     solver='adam',
     random_state=1)
     }

    c_d16 = mlpc
    classifiers.update(c_d16)

    cols = ['model', 'run_time', 'roc_auc']
    df_models = pd.DataFrame(columns=cols)

    for key in classifiers:
        start_time = time.time()

        pipeline = model_pipeline(x_train, classifiers[key])

        cv = cross_val_score(pipeline, x, y, cv=10, scoring='roc_auc')

        row = {'model': key, 'run_time': format(round((time.time() - start_time)

        df_models = pd.concat([df_models, pd.DataFrame([row])], ignore_index=Tru

    df_models = df_models.sort_values(by='roc_auc', ascending=False)

    return df_models
```

In [84]:
```
models = select_model(x_train,y_train)
```

```
[LightGBM] [Info] Number of positive: 6291, number of negative: 6291
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.000800 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1315
[LightGBM] [Info] Number of data points in the train set: 12582, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 6292, number of negative: 6292
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.000379 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1066
[LightGBM] [Info] Number of data points in the train set: 12584, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 6292, number of negative: 6292
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.000301 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1062
[LightGBM] [Info] Number of data points in the train set: 12584, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 6292, number of negative: 6292
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.000705 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1315
[LightGBM] [Info] Number of data points in the train set: 12584, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 6292, number of negative: 6292
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.000866 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1316
[LightGBM] [Info] Number of data points in the train set: 12584, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 6292, number of negative: 6292
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.000598 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1317
[LightGBM] [Info] Number of data points in the train set: 12584, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 6292, number of negative: 6292
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.000350 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1067
[LightGBM] [Info] Number of data points in the train set: 12584, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
```

```
[LightGBM] [Info] Number of positive: 6292, number of negative: 6292
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.000712 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1318
[LightGBM] [Info] Number of data points in the train set: 12584, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 6292, number of negative: 6292
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.000171 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1315
[LightGBM] [Info] Number of data points in the train set: 12584, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 6292, number of negative: 6292
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.001099 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1315
[LightGBM] [Info] Number of data points in the train set: 12584, number of used f
eatures: 6
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
```

time diaplay

In [85]:

```
models
```

Out[85]:

	model	run_time	roc_auc
15	MLPClassifier (paper)	1.74	0.907164
6	lightgbm	0.03	0.903916
2	Xgboost	0.04	0.895602
3	RandomForestClassifier	0.39	0.895006
10	extratrees	0.21	0.890312
5	svc	0.89	0.889579
13	SVC	0.87	0.889579
12	SGDC	0.02	0.888550
14	BernoulliNB	0.01	0.861823
11	redge	0.01	0.855420
7	kneighbors	0.02	0.842293
8	KNeighborsClassifier	0.02	0.842293
9	extratree	0.02	0.761791
4	DecisionTreeClassifier	0.03	0.759974
0	DummyClassifier	0.01	0.500000
1	linearRegression	0.01	NaN

In [ ]:

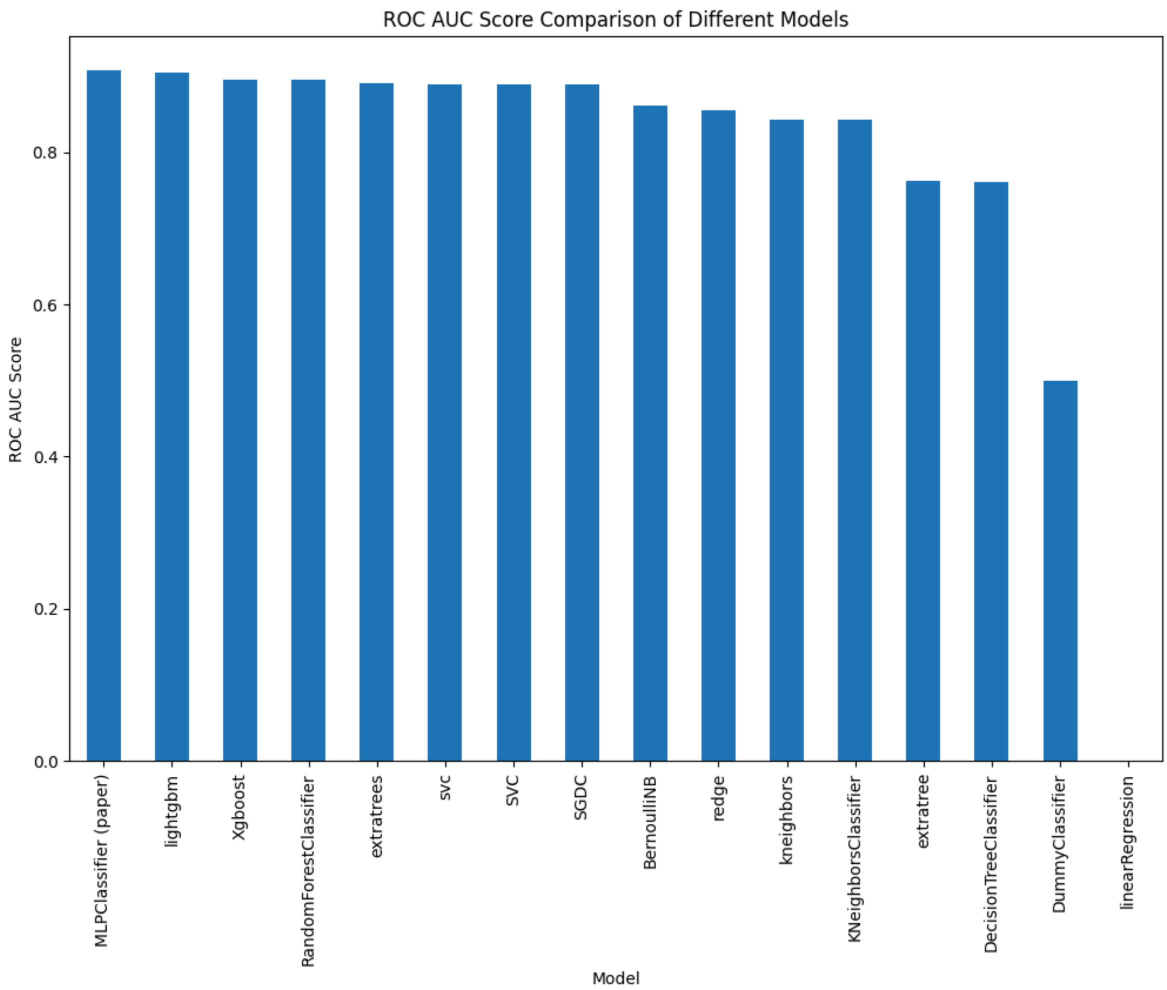
In [120…

```
models.plot.bar(x='model', y='roc_auc',legend=False,figsize=(12,8))

# Adding some titles and labels
plt.title('ROC AUC Score Comparison of Different Models')
plt.xlabel('Model')
plt.ylabel('ROC AUC Score')
```

Out[120…

```
Text(0, 0.5, 'ROC AUC Score')
```



In [124…

```
models.sum()
```

Out[124…

```
model       MLPClassifier (paper)lightgbmXgboostRandomFore...
run_time    1.740.030.040.390.210.890.870.020.010.010.020....
roc_auc                                             12.583303
dtype: object
```

Model Creation

In [121…

```
selected_model = MLPClassifier()
```

In [87]:

```
select_model
```

Out[87]:

```
<function __main__.select_model(x, y, pipeline=None)>
```

In [88]:

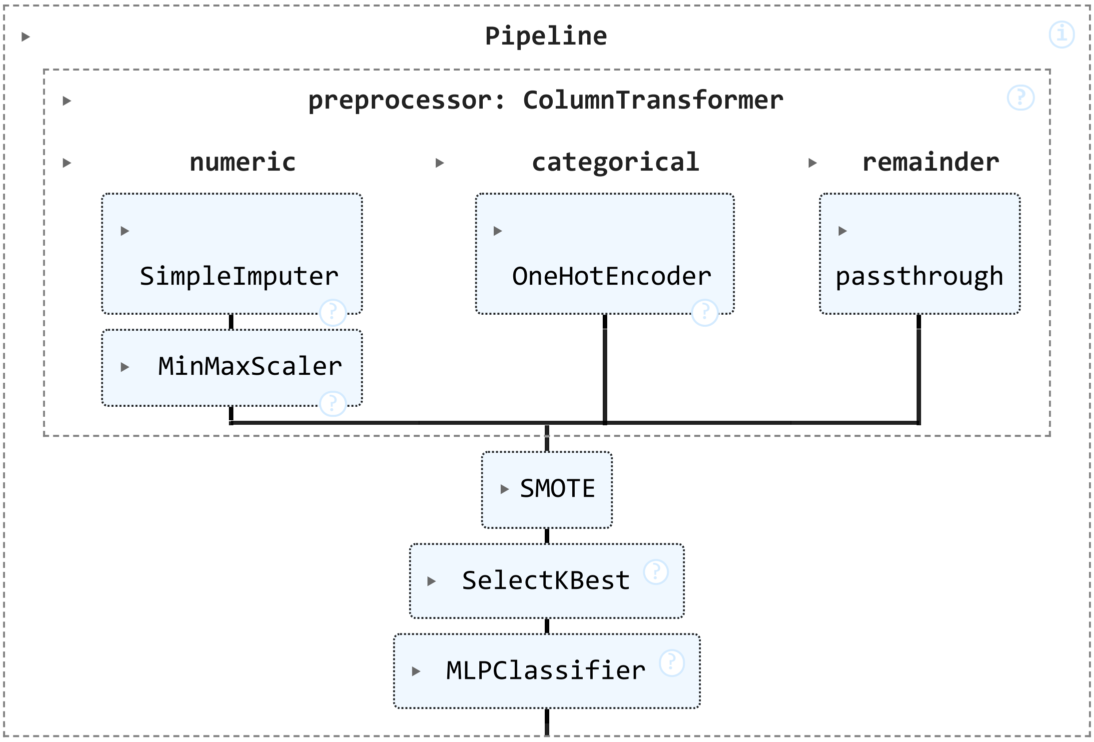
```
bundled_pipeline = model_pipeline(x_train, selected_model)
```

model training

In [89]:

```
bundled_pipeline.fit(x_train, y_train)
```

Out[89]:



Prediction

In [62]:
```
y_pred = bundled_pipeline.predict(x_test)
```

In [90]:
```
y_pred
```

Out[90]:
```
array([0, 0, 0, ..., 0, 1, 0])
```

In [91]:
```
roc_auc = roc_auc_score(y_test, y_pred)
```

In [92]:
```
roc_auc
```

Out[92]:
```
np.float64(0.838457261790662)
```

In [93]:
```
accuracy = accuracy_score(y_test, y_pred)
```

In [94]:
```
accuracy
```

Out[94]:
```
0.8771196854263947
```

Classicication-report

In [105…
```
classif_report = classification_report(y_test, y_pred)
```

In [107…
```
print(classif_report)
```

```
              precision    recall  f1-score   support

           0       0.96      0.89      0.92      3431
           1       0.58      0.78      0.67       638

    accuracy                           0.88      4069
   macro avg       0.77      0.84      0.80      4069
weighted avg       0.90      0.88      0.88      4069
```

In [ ]: