```
## Prepaid Telecom Datasets
```

About Dataset

Lack of such data on kaggle inspired me to create this dataset as people can use to analyze and evaluate these plans and create necessary visualizations too.
This is cited from the official websites of VI and Airtel. This is my first dataset so any kind of criticism is welcome

Shares all the unlimited call plans offered by VI and AIrtel as of 16th July 2022

The features are as follows:
SIM_COMPANY : Specifies VI or Airtel

OFFER-PRICE(INR) : Price of the offer

VALIDITY(DAYS) : Plan Validity

DATA-PER-DAY(GB) : Internet Data provided on daily basis

ADDITIONAL-DATA(GB) : Data provided as whole with the pack

SMS-PER-DAY : Number of Free SMS on daily basis

ADDITIONAL-SMS : Free SMS included with the pack

DISNEY+HOTSTAR(Months) : Number of months for Free subscription of Disney/Hotstar

COST-PER-DAY : Cost calculation based on plan validity and price offered

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

```
# Importing Pandas and NumPy
import pandas as pd, numpy as np
```

```
###!mkdir ~/.kaggle
```

```
###!cp /kaggle.json ~/.kaggle/
```

```
###!chmod 600 ~/.kaggle/kaggle.json
```

```
####! pip install kaggle
```

```
###! kaggle datasets download -d abdulaziz04/vi-and-airtel-prepaid-plans-dataset
```

```
###! unzip /content/vi-and-airtel-prepaid-plans-dataset.zip
```

```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

```
# Importing all datasets
vi_airtel_plans = pd.read_csv("/content/VI_AIRTEL_PLANS.csv")
vi_airtel_plans.head(4)
```

	SIM_COMPANY	OFFER_PRICE	VALIDITY	DATA_PER_DAY	ADDITIONAL DATA	SMS_PER_DAY	ADDITIONAL_SMS	DISNEY+HOTSTAR	COST_PER_DAY
0	AIRTEL	455	84	0.0	6.0	0	900	0	5.42
1	AIRTEL	299	28	2.0	0.0	100	0	0	10.68
2	AIRTEL	479	56	1.5	0.0	100	0	0	8.55
3	AIRTEL	155	24	0.0	1.0	0	300	0	6.46

```
vi_airtel_plans.dtypes
```

```
SIM_COMPANY         object
OFFER_PRICE          int64
VALIDITY             int64
DATA_PER_DAY       float64
ADDITIONAL DATA    float64
SMS_PER_DAY          int64
ADDITIONAL_SMS       int64
DISNEY+HOTSTAR       int64
COST_PER_DAY       float64
dtype: object
```

```
vi_airtel_plans.shape
```

```
(55, 9)
```

```
vi_airtel_plans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55 entries, 0 to 54
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   SIM_COMPANY      55 non-null     object
 1   OFFER_PRICE      55 non-null     int64
 2   VALIDITY         55 non-null     int64
 3   DATA_PER_DAY     55 non-null     float64
 4   ADDITIONAL DATA  55 non-null     float64
 5   SMS_PER_DAY      55 non-null     int64
 6   ADDITIONAL_SMS   55 non-null     int64
 7   DISNEY+HOTSTAR   55 non-null     int64
 8   COST_PER_DAY     55 non-null     float64
dtypes: float64(3), int64(5), object(1)
memory usage: 4.0+ KB
```

```
vi_airtel_plans.columns
```

```
Index(['SIM_COMPANY', 'OFFER_PRICE', 'VALIDITY', 'DATA_PER_DAY',
       'ADDITIONAL DATA', 'SMS_PER_DAY', 'ADDITIONAL_SMS', 'DISNEY+HOTSTAR',
       'COST_PER_DAY'],
      dtype='object')
```

```
vi_airtel_plans["SIM_COMPANY"] = vi_airtel_plans["SIM_COMPANY"].astype("category").cat.codes
```

```
vi_airtel_plans.head(3)
```

	SIM_COMPANY	OFFER_PRICE	VALIDITY	DATA_PER_DAY	ADDITIONAL DATA	SMS_PER_DAY	ADDITIONAL_SMS	DISNEY+HOTSTAR	COST_PER_DAY
0	0	455	84	0.0	6.0	0	900	0	5.42
1	0	299	28	2.0	0.0	100	0	0	10.68
2	0	479	56	1.5	0.0	100	0	0	8.55

```
vi_airtel_plans["SIM_COMPANY"].value_counts()
```

```
1    39
0    16
Name: SIM_COMPANY, dtype: int64
```

```
vi_airtel_plans.columns
```

```
Index(['SIM_COMPANY', 'OFFER_PRICE', 'VALIDITY', 'DATA_PER_DAY',
       'ADDITIONAL DATA', 'SMS_PER_DAY', 'ADDITIONAL_SMS', 'DISNEY+HOTSTAR',
       'COST_PER_DAY'],
      dtype='object')
```

```
X = vi_airtel_plans.drop(['OFFER_PRICE'], axis=1)
```

```
Y = vi_airtel_plans["OFFER_PRICE"]
```

```
from sklearn.ensemble import ExtraTreesRegressor
import matplotlib.pyplot as plt
model=ExtraTreesRegressor()
model.fit(X,Y)
```

```
▾ ExtraTreesRegressor
ExtraTreesRegressor()
```

```
print(model.feature_importances_)
```

```
[8.61568030e-05 8.70804305e-01 2.75957752e-02 1.22201590e-02
 2.59117873e-02 1.18250246e-02 1.49170220e-02 3.66397705e-02]
```

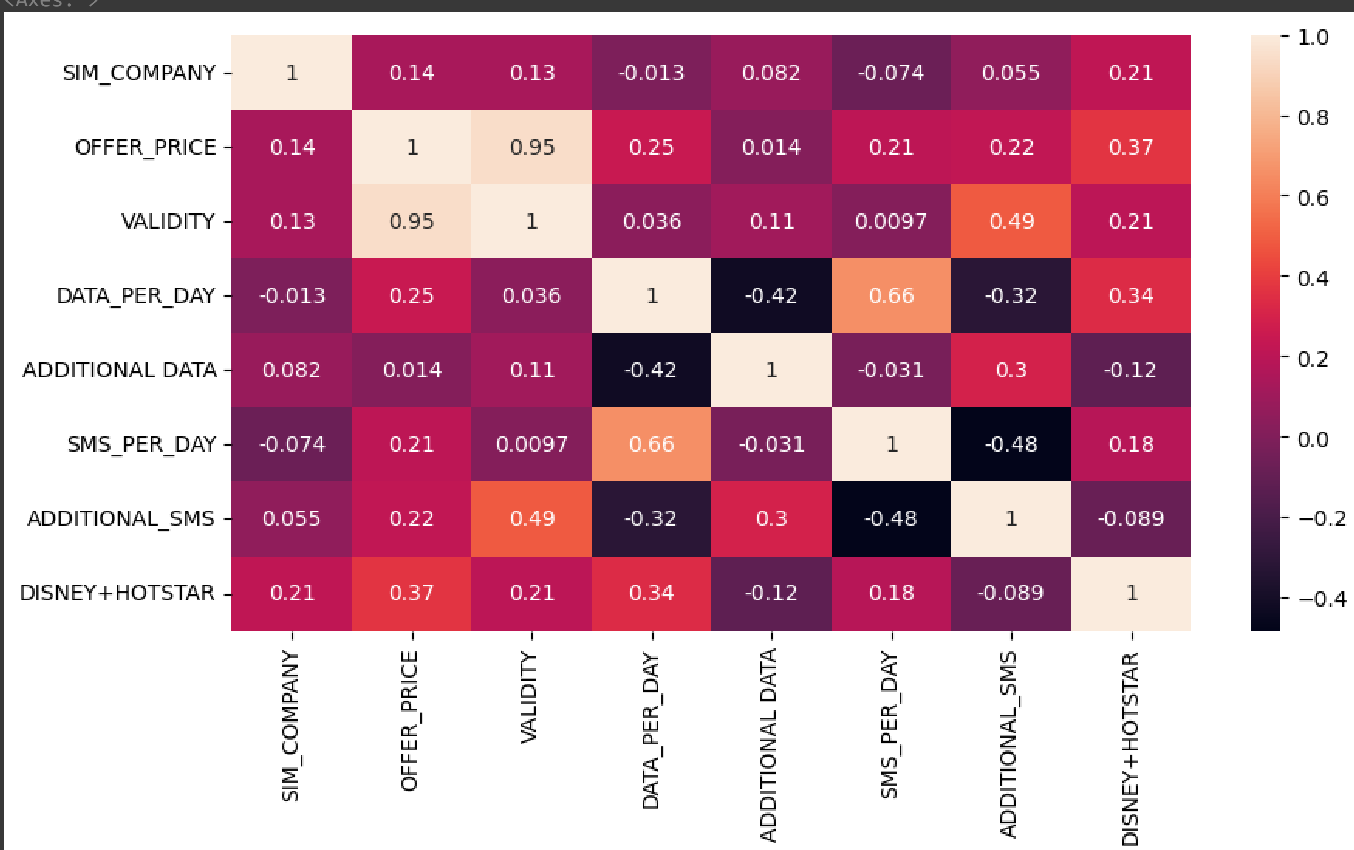
```
ranked_features=pd.Series(model.feature_importances_,index=X.columns)
ranked_features.nlargest(10).plot(kind='barh')
plt.show()
```

```
vi_airtel_plans.corr()
```

	SIM_COMPANY	OFFER_PRICE	VALIDITY	DATA_PER_DAY	ADDITIONAL DATA	SMS_PER_DAY	ADDITIONAL_SMS	DISNEY+HOTSTAR	COST
SIM_COMPANY	1.000000	0.136162	0.132975	-0.012639	0.082158	-0.073668	0.055370	0.212304	
OFFER_PRICE	0.136162	1.000000	0.946479	0.249550	0.013719	0.208047	0.221872	0.369923	
VALIDITY	0.132975	0.946479	1.000000	0.035517	0.111464	0.009699	0.485600	0.205497	
DATA_PER_DAY	-0.012639	0.249550	0.035517	1.000000	-0.424601	0.656389	-0.317188	0.337425	
ADDITIONAL DATA	0.082158	0.013719	0.111464	-0.424601	1.000000	-0.031143	0.298840	-0.119288	
SMS_PER_DAY	-0.073668	0.208047	0.009699	0.656389	-0.031143	1.000000	-0.483231	0.184407	
ADDITIONAL_SMS	0.055370	0.221872	0.485600	-0.317188	0.298840	-0.483231	1.000000	-0.089111	
DISNEY+HOTSTAR	0.212304	0.369923	0.205497	0.337425	-0.119288	0.184407	-0.089111	1.000000	
COST_PER_DAY	0.075269	-0.032547	-0.248447	0.764235	-0.136094	0.624858	-0.375466	0.502491	

```
import seaborn as sns
corr=vi_airtel_plans.iloc[:,:-1].corr()
top_features=corr.index
plt.figure(figsize=(10,5))
sns.heatmap(vi_airtel_plans[top_features].corr(),annot=True)
```

<Axes: >




```
#### Remove The correlated
```

```
threshold=0.8
```

```
# find and remove correlated features
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

```
correlation(vi_airtel_plans.iloc[:,:-1],threshold)
```

```
{'VALIDITY'}
```

```
X = X.drop(['VALIDITY'], axis=1)
```

```
X.shape
```

```
(55, 7)
```

INFORMATION GAIN

```
from sklearn.feature_selection import mutual_info_regression
```

```
mutual_info=mutual_info_regression(X,Y)
```

```
mutual_data=pd.Series(mutual_info,index=X.columns)
mutual_data.sort_values(ascending=False)
```

```
COST_PER_DAY       0.559773
DATA_PER_DAY       0.449937
SMS_PER_DAY        0.261368
ADDITIONAL_SMS     0.219533
ADDITIONAL DATA    0.179127
DISNEY+HOTSTAR     0.025779
SIM_COMPANY        0.000000
dtype: float64
```

```
X =  X.drop(['SIM_COMPANY'], axis=1)
```

```
# columnslitting the data into train and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7, test_size=0.3, random_state=100)
```

```
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(38, 6)
(17, 6)
(38,)
(17,)
```

```
## Pipelines Creation
## 1. Data Preprocessing by using Standard Scaler
## 2. Reduce Dimension using PCA
## 3. Apply  Classifier
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
```

```
pipeline_lr=Pipeline([('scalar1',StandardScaler()),
                     ('pca1',PCA(n_components=5)),
                     ('lin_regressor',LinearRegression())])
```

```
pipeline_dt=Pipeline([('scalar2',StandardScaler()),
                     ('pca2',PCA(n_components=5)),
                     ('dt_regressor',DecisionTreeRegressor())])
```

```
pipeline_randomforest=Pipeline([('scalar3',StandardScaler()),
                     ('pca3',PCA(n_components=5)),
                     ('rf_classifier',RandomForestRegressor())])
```

```
pipeline_xgbregressor=Pipeline([('scalar3',StandardScaler()),
                     ('pca3',PCA(n_components=5)),
                     ('rf_classifier',XGBRegressor())])
```

```
## LEts make the list of pipelines
pipelines = [pipeline_lr, pipeline_dt, pipeline_randomforest, pipeline_xgbregressor]
```

```
best_accuracy=0.0
best_regressor=0
best_pipeline=""
```

```
# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'RandomForest', 3: 'XGBRegressor'}

# Fit the pipelines
for pipe in pipelines:
    pipe.fit(X_train, Y_train)
```

```
for i,model in enumerate(pipelines):
    print("{} Test Accuracy: {}".format(pipe_dict[i],model.score(X_test,Y_test)))
```

```
Logistic Regression Test Accuracy: -0.4561916257277665
Decision Tree Test Accuracy: -0.9278459071578158
RandomForest Test Accuracy: -0.2747418384630762
XGBRegressor Test Accuracy: -0.028427853192655617
```

```
xgb_regressor=XGBRegressor(random_state=0).fit(X_train,Y_train)
prediction=xgb_regressor.predict(X_test)
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error, r2_score
print("Mean Squarred Error : ",mean_squared_error(Y_test,prediction))
print("Mean Absolute Error : ",mean_absolute_error(Y_test,prediction))
print("Mean Absolute Percentage Error : ",mean_absolute_percentage_error(Y_test,prediction))
```

```
Mean Squarred Error :  124489.21692058578
Mean Absolute Error :  160.24459120806526
Mean Absolute Percentage Error :  0.340474664135147
```

```
print("R2 Score : ",r2_score(Y_test,prediction))
```

```
R2 Score :  0.10214769286962355
```

```
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import time
```

Randomized Search CV

```
# A parameter grid for XGBoost
params = {
    'n_estimators':[500],
    'min_child_weight':[4,5],
    'gamma':[i/10.0 for i in range(3,6)],
    'subsample':[i/10.0 for i in range(6,11)],
    'colsample_bytree':[i/10.0 for i in range(6,11)],
    'max_depth': [2,3,4,6,7],
    'objective': ['reg:squarederror', 'reg:tweedie'],
    'booster': ['gbtree', 'gblinear'],
    'eval_metric': ['rmse'],
    'eta': [i/10.0 for i in range(3,6)],
}

reg = XGBRegressor(nthread=-1)

# run randomized search
n_iter_search = 500
random_search = RandomizedSearchCV(reg, param_distributions=params,
                                   n_iter=n_iter_search, cv=9, scoring='neg_mean_squared_error')

start = time.time()
random_search.fit(X_train, Y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
```

```
[17:53:52] WARNING: ../src/learner.cc:767:
Parameters: { "colsample_bytree", "gamma", "max_depth", "min_child_weight", "subsample" } are not used.

[17:53:52] WARNING: ../src/learner.cc:767:
Parameters: { "colsample_bytree", "gamma", "max_depth", "min_child_weight", "subsample" } are not used.

[17:53:52] WARNING: ../src/learner.cc:767:
Parameters: { "colsample_bytree", "gamma", "max_depth", "min_child_weight", "subsample" } are not used.

[17:53:52] WARNING: ../src/learner.cc:767:
Parameters: { "colsample_bytree", "gamma", "max_depth", "min_child_weight", "subsample" } are not used.

[17:53:52] WARNING: ../src/learner.cc:767:
Parameters: { "colsample_bytree", "gamma", "max_depth", "min_child_weight", "subsample" } are not used.

[17:53:52] WARNING: ../src/learner.cc:767:
Parameters: { "colsample_bytree", "gamma", "max_depth", "min_child_weight", "subsample" } are not used.

[17:53:52] WARNING: ../src/learner.cc:767:
Parameters: { "colsample_bytree", "gamma", "max_depth", "min_child_weight", "subsample" } are not used.

[17:53:55] WARNING: ../src/learner.cc:767:
Parameters: { "colsample_bytree", "gamma", "max_depth", "min_child_weight", "subsample" } are not used.

RandomizedSearchCV took 395.15 seconds for 500 candidates parameter settings.
```

```
best_regressor = random_search.best_estimator_
```

```
print(best_regressor)
```

```
XGBRegressor(base_score=None, booster='gblinear', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.7, early_stopping_rounds=None,
             enable_categorical=False, eta=0.5, eval_metric='rmse',
             feature_types=None, gamma=0.5, gpu_id=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=6,
             max_leaves=None, min_child_weight=5, missing=nan,
             monotone_constraints=None, n_estimators=500, n_jobs=None,
             nthread=-1, num_parallel_tree=None, ...)
```

```
y_pred_rmse=best_regressor.predict(X_test)
```

```
y_pred_rmse[:10]
```

```
array([176.53297, 802.28906, 458.02164, 458.02164, 275.22815, 346.73532,
       430.7942 , 519.7985 , 500.438  , 561.88245], dtype=float32)
```

```
print("Mean Squarred Error : ",mean_squared_error(Y_test,y_pred_rmse))
print("Mean Absolute Error : ",mean_absolute_error(Y_test,y_pred_rmse))
print("Mean Absolute Percentage Error : ",mean_absolute_percentage_error(Y_test,y_pred_rmse))
```

```
Mean Squarred Error :  1940808.1379003136
Mean Absolute Error :  500.1948592242073
Mean Absolute Percentage Error :  0.7235975106857594
```

```
print("R2 Score : ",r2_score(Y_test,y_pred_rmse))
```

```
R2 Score :  -12.997670701252948
```

```
random_search.best_params_
```

```
{'subsample': 1.0,
 'objective': 'reg:tweedie',
 'n_estimators': 500,
 'min_child_weight': 5,
 'max_depth': 6,
 'gamma': 0.5,
```

```
 'eval_metric': 'rmse',
 'eta': 0.5,
 'colsample_bytree': 0.7,
 'booster': 'gblinear'}
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'min_child_weight': [random_search.best_params_['min_child_weight']],
    'n_estimators': [random_search.best_params_['n_estimators'],
                     random_search.best_params_['n_estimators']+200,
                     random_search.best_params_['n_estimators']+300,
                     random_search.best_params_['n_estimators']+400,
                     random_search.best_params_['n_estimators']+500],
    'max_depth': [random_search.best_params_['max_depth'],
                         random_search.best_params_['max_depth']+2,
                         random_search.best_params_['max_depth'] + 4,
                         random_search.best_params_['max_depth'] + 8,
                         random_search.best_params_['max_depth'] + 12],
    'min_child_weight': [random_search.best_params_['min_child_weight'] - 2,
                          random_search.best_params_['min_child_weight'] - 1,
                          random_search.best_params_['min_child_weight'],
                          random_search.best_params_['min_child_weight'] + 1,
                          random_search.best_params_['min_child_weight'] + 2]
}

print(param_grid)
```

```
{'min_child_weight': [3, 4, 5, 6, 7], 'n_estimators': [500, 700, 800, 900, 1000], 'max_depth': [6, 8, 10, 14, 18]}
```

```
#### Fit the grid_search to the data
xgb=XGBRegressor()
grid_search=GridSearchCV(estimator=xgb,param_grid=param_grid,cv=10,n_jobs=-1,verbose=2)
grid_search.fit(X_train,Y_train)
```

```
Fitting 10 folds for each of 125 candidates, totalling 1250 fits
```

GridSearchCV

estimator: XGBRegressor

XGBRegressor

```
grid_search.best_estimator_
```

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=8, max_leaves=None,
             min_child_weight=3, missing=nan, monotone_constraints=None,
             n_estimators=500, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

```
best_grid=grid_search.best_estimator_
```

```
best_grid.fit(X_train, Y_train)
```

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
```

```
y_pred_grid = best_grid.predict(X_test)
```

```
             n_estimators=500, n_jobs=None, num_parallel_tree=None,
```

```
y_pred_grid[:10]
```

```
array([ 900.4427  ,  666.03705 ,  209.0006  ,  209.0006  ,  337.0012  ,
         52.109844,  458.8447  ,  -35.850918,  -35.850918, 1089.7617  ],
      dtype=float32)
```

```
print("Mean Squarred Error : ",mean_squared_error(Y_test,y_pred_grid))
print("Mean Absolute Error : ",mean_absolute_error(Y_test,y_pred_grid))
print("Mean Absolute Percentage Error : ",mean_absolute_percentage_error(Y_test,y_pred_grid))
```

```
Mean Squarred Error :  176466.2675812582
Mean Absolute Error :  230.1328737595502
Mean Absolute Percentage Error :  0.6066290022471118
```

Optuna

```
###! pip install optuna
```

```
Installing collected packages: Mako, colorlog, cmaes, alembic, optuna
Successfully installed Mako-1.2.4 alembic-1.11.1 cmaes-0.10.0 colorlog-6.7.0 optuna-3.2.0
```

```
###! pip install xgboost
```

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
import optuna
```

```
def objective(trial):
    params = {
        "objective": "reg:tweedie",
        "n_estimators": trial.suggest_int("n_estimators", 1000, 2000),
        "verbosity": 0,
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.1, log=True),
        "max_depth": trial.suggest_int("max_depth", 1, 10),
        "subsample": trial.suggest_float("subsample", 0.05, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.05, 1.0),
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 100),
    }

    model = xgb.XGBRegressor(**params)
    model.fit(X_train, Y_train, verbose=False)
    predictions_optuna = model.predict(X_test)
    rmse = mean_squared_error(Y_test, predictions_optuna, squared=False)
    return rmse
```

```
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=300)
```

```
[I 2023-07-30 18:01:11,510] Trial 247 finished with value: 262.1843633886421 and parameters: {'n_estimators': 1904, 'learning_
[I 2023-07-30 18:01:11,887] Trial 248 finished with value: 130.35999611926817 and parameters: {'n_estimators': 1938, 'learning
[I 2023-07-30 18:01:12,245] Trial 249 finished with value: 117.54411411700836 and parameters: {'n_estimators': 1885, 'learning
[I 2023-07-30 18:01:12,628] Trial 250 finished with value: 365.94417123172514 and parameters: {'n_estimators': 1845, 'learning
[I 2023-07-30 18:01:12,991] Trial 251 finished with value: 93.31154331626375 and parameters: {'n_estimators': 1869, 'learning_
[I 2023-07-30 18:01:13,340] Trial 252 finished with value: 510.1984182201119 and parameters: {'n_estimators': 1867, 'learning_
[I 2023-07-30 18:01:13,722] Trial 253 finished with value: 274.6572601882892 and parameters: {'n_estimators': 1905, 'learning_
[I 2023-07-30 18:01:14,065] Trial 254 finished with value: 223.88920787673845 and parameters: {'n_estimators': 1827, 'learning
[I 2023-07-30 18:01:14,431] Trial 255 finished with value: 398.26343620570475 and parameters: {'n_estimators': 1875, 'learning
[I 2023-07-30 18:01:14,802] Trial 256 finished with value: 303.2429489094438 and parameters: {'n_estimators': 1854, 'learning_
[I 2023-07-30 18:01:17,863] Trial 257 finished with value: 153.93573156006386 and parameters: {'n_estimators': 1833, 'learning
[I 2023-07-30 18:01:18,319] Trial 258 finished with value: 335.80830914882824 and parameters: {'n_estimators': 1890, 'learning
[I 2023-07-30 18:01:18,666] Trial 259 finished with value: 236.4770576973163 and parameters: {'n_estimators': 1913, 'learning_
[I 2023-07-30 18:01:19,051] Trial 260 finished with value: 85.84219011204685 and parameters: {'n_estimators': 1871, 'learning_
[I 2023-07-30 18:01:19,399] Trial 261 finished with value: 364.3428524582104 and parameters: {'n_estimators': 1872, 'learning_
[I 2023-07-30 18:01:19,797] Trial 262 finished with value: 321.50634333220705 and parameters: {'n_estimators': 1857, 'learning
[I 2023-07-30 18:01:20,160] Trial 263 finished with value: 124.15923284113373 and parameters: {'n_estimators': 1891, 'learning
[I 2023-07-30 18:01:20,522] Trial 264 finished with value: 394.95503724989754 and parameters: {'n_estimators': 1845, 'learning
[I 2023-07-30 18:01:20,915] Trial 265 finished with value: 371.83844460062954 and parameters: {'n_estimators': 1887, 'learning
[I 2023-07-30 18:01:21,277] Trial 266 finished with value: 101.89128959754888 and parameters: {'n_estimators': 1868, 'learning
[I 2023-07-30 18:01:21,661] Trial 267 finished with value: 304.48821508386703 and parameters: {'n_estimators': 1821, 'learning
[I 2023-07-30 18:01:22,045] Trial 268 finished with value: 390.59383013963395 and parameters: {'n_estimators': 1902, 'learning
[I 2023-07-30 18:01:22,411] Trial 269 finished with value: 302.0115866299114 and parameters: {'n_estimators': 1854, 'learning_
[I 2023-07-30 18:01:22,790] Trial 270 finished with value: 423.9380360462109 and parameters: {'n_estimators': 1925, 'learning_
[I 2023-07-30 18:01:23,147] Trial 271 finished with value: 343.40873078994474 and parameters: {'n_estimators': 1829, 'learning
[I 2023-07-30 18:01:23,530] Trial 272 finished with value: 334.19626930418866 and parameters: {'n_estimators': 1874, 'learning
[I 2023-07-30 18:01:23,899] Trial 273 finished with value: 316.9104031058097 and parameters: {'n_estimators': 1907, 'learning_
[I 2023-07-30 18:01:24,256] Trial 274 finished with value: 366.5003050369052 and parameters: {'n_estimators': 1846, 'learning_
[I 2023-07-30 18:01:24,556] Trial 275 finished with value: 145.05286579510704 and parameters: {'n_estimators': 1347, 'learning
[I 2023-07-30 18:01:24,934] Trial 276 finished with value: 321.1773809311424 and parameters: {'n_estimators': 1961, 'learning_
[I 2023-07-30 18:01:25,289] Trial 277 finished with value: 217.64654856874674 and parameters: {'n_estimators': 1810, 'learning
[I 2023-07-30 18:01:25,646] Trial 278 finished with value: 288.47220452160605 and parameters: {'n_estimators': 1880, 'learning
[I 2023-07-30 18:01:26,031] Trial 279 finished with value: 381.24780088610703 and parameters: {'n_estimators': 1862, 'learning
[I 2023-07-30 18:01:26,303] Trial 280 finished with value: 372.23855829550155 and parameters: {'n_estimators': 1240, 'learning
[I 2023-07-30 18:01:26,677] Trial 281 finished with value: 374.023353867601 and parameters: {'n_estimators': 1940, 'learning_r
[I 2023-07-30 18:01:27,112] Trial 282 finished with value: 327.28595250018003 and parameters: {'n_estimators': 1893, 'learning
[I 2023-07-30 18:01:27,477] Trial 283 finished with value: 144.49012558867898 and parameters: {'n_estimators': 1790, 'learning
[I 2023-07-30 18:01:28,074] Trial 284 finished with value: 94.4959238474741 and parameters: {'n_estimators': 1916, 'learning_r
[I 2023-07-30 18:01:30,419] Trial 285 finished with value: 351.9448227387416 and parameters: {'n_estimators': 1920, 'learning_
[I 2023-07-30 18:01:30,768] Trial 286 finished with value: 180.39391405524236 and parameters: {'n_estimators': 1843, 'learning
[I 2023-07-30 18:01:31,135] Trial 287 finished with value: 332.09496887575386 and parameters: {'n_estimators': 1916, 'learning
[I 2023-07-30 18:01:31,520] Trial 288 finished with value: 199.07577983309858 and parameters: {'n_estimators': 1949, 'learning
[I 2023-07-30 18:01:31,891] Trial 289 finished with value: 293.4454773811434 and parameters: {'n_estimators': 1894, 'learning_
[I 2023-07-30 18:01:32,267] Trial 290 finished with value: 90.08315906423431 and parameters: {'n_estimators': 1865, 'learning_
[I 2023-07-30 18:01:32,662] Trial 291 finished with value: 272.25579149642874 and parameters: {'n_estimators': 1998, 'learning
[I 2023-07-30 18:01:33,052] Trial 292 finished with value: 369.46057730764375 and parameters: {'n_estimators': 1873, 'learning
[I 2023-07-30 18:01:34,156] Trial 293 finished with value: 375.83034821796304 and parameters: {'n_estimators': 1907, 'learning
[I 2023-07-30 18:01:34,655] Trial 294 finished with value: 260.15126382691955 and parameters: {'n_estimators': 1880, 'learning
[I 2023-07-30 18:01:35,532] Trial 295 finished with value: 403.1730242723328 and parameters: {'n_estimators': 1932, 'learning_
[I 2023-07-30 18:01:35,894] Trial 296 finished with value: 307.2561914081533 and parameters: {'n_estimators': 1869, 'learning_
[I 2023-07-30 18:01:36,280] Trial 297 finished with value: 397.46501155787945 and parameters: {'n_estimators': 1896, 'learning
[I 2023-07-30 18:01:36,643] Trial 298 finished with value: 379.1528459166167 and parameters: {'n_estimators': 1854, 'learning_
[I 2023-07-30 18:01:37,013] Trial 299 finished with value: 300.87570806204786 and parameters: {'n_estimators': 1816, 'learning
```

```python
print('Best hyperparameters:', study.best_params)
print('Best RMSE:', study.best_value)
```

```
Best hyperparameters: {'n_estimators': 1792, 'learning_rate': 0.0979511214309094, 'max_depth': 1, 'subsample': 0.07882167899833
Best RMSE: 85.46594064602196
```

```python
xgboost_regressor = xgb.XGBRegressor(**study.best_params)
```

```python
xgboost_regressor.fit(X_train, Y_train)
```

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.6931231677868107, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
```

```
y_pred_xgboost = xgboost_regressor.predict(X_test)
```

```
             max_delta_step=None, max_depth=1, max_leaves=None,
```

```
y_pred_xgboost[:10]
```

```
array([734.009  , 612.48126, 569.92236, 569.92236, 560.4196 , 579.4037 ,
       593.6781 , 521.59454, 521.59454, 593.6781 ], dtype=float32)
```

```
print("Mean Squarred Error : ",mean_squared_error(Y_test,y_pred_xgboost))
print("Mean Absolute Error : ",mean_absolute_error(Y_test,y_pred_xgboost))
print("Mean Absolute Percentage Error : ",mean_absolute_percentage_error(Y_test,y_pred_xgboost))
```

```
Mean Squarred Error :  151234.14036592183
Mean Absolute Error :  293.1332648782169
Mean Absolute Percentage Error :  0.8336880538827112
```

```
print("R2 Score : ",r2_score(Y_test,y_pred_xgboost))
```

```
R2 Score :  -0.0907444452079953
```

```
y_pred_xgboost[:10]
```

```
array([734.009  , 612.48126, 569.92236, 569.92236, 560.4196 , 579.4037 ,
       593.6781 , 521.59454, 521.59454, 593.6781 ], dtype=float32)
```

```
X_test.columns
```

```
Index(['DATA_PER_DAY', 'ADDITIONAL DATA', 'SMS_PER_DAY', 'ADDITIONAL_SMS',
       'DISNEY+HOTSTAR', 'COST_PER_DAY'],
      dtype='object')
```

```
y_pred_xgboost = pd.DataFrame(y_pred_xgboost)
```

```
y_pred_xgboost.rename(columns = { 0 : "PREDICT"}, inplace=True)
```

```
frames = [y_pred_xgboost, X_test]
```

```
res1 = pd.concat([y_pred_xgboost, X_test], axis=1, join='inner')
```

```
res1.head(2)
```

	PREDICT	DATA_PER_DAY	ADDITIONAL DATA	SMS_PER_DAY	ADDITIONAL_SMS	DISNEY+HOTSTAR	COST_PER_DAY
0	734.008972	0.0	6.0	0	900	0	5.42
5	579.403687	1.5	0.0	100	0	0	8.56

```
res1.to_csv("Predict_Price.csv")
```