Topic: Content monetisation & Revenue prediction on YouTube data.

YouTube is a highly popular platform for creators of video content, with more than 2 billion monthly active users. As a result, it has become a popular destination for content creators to showcase their abilities and reach a larger audience. Nonetheless, with such an immense amount of content available, it can be difficult to get noticed and obtain the necessary views and subscribers to monetise the content that they are posting. To overcome these challenges, YouTubers must optimize their videos to obtain the maximum amount of engagement, views, and subscribers. The dataset provided below takes into consideration various features that play a pivotal role in determining the revenue of Youtubers.

Colab link: https://colab.research.google.com/drive/1XK2f3vBINhJI9nW5d2VMgrIy4TygpHRK#scrollTo=pLUgLP6w_RRw

Removing outliers & splitting the data into training & testing sets.

```
[ ]  def remove_outliers(df, col_list, z_thresh=4.3):
         for col in col_list:
             z_scores = np.abs((df[col] - df[col].mean()) / df[col].std())
             df = df[z_scores < z_thresh]
         return df
     df=remove_outliers(df,columns,4.3)
```

Removing the entries where the year is 2018 since the revenue in 2018 is negigible

```
[ ]  df=df[df['year']!=2018]
```

```
[ ]  len(df)
```

```
885
```

Converting average view duration from object datatype to seconds:

```
[ ]  df['Average view duration']=pd.to_timedelta(df['Average view duration']).dt.total_seconds()
```

Splitting the data into training & testing with respect to a particular date:

> Since it is a time series data, performing a random split would not be the appropriate choice in this case.

```
[ ]  x_train=df[df['Date']<'01-10-2021'].drop(columns=['Date','Your estimated revenue (USD)'],axis=1).reset_index(drop=True)
     y_train=df[df['Date']<'01-10-2021'].drop(columns='Date',axis=1)['Your estimated revenue (USD)'].reset_index(drop=True)
     x_test=df[df['Date']>='01-10-2021'].drop(columns=['Date','Your estimated revenue (USD)'],axis=1).reset_index(drop=True)
     y_test=df[df['Date']>='01-10-2021'].drop(columns='Date',axis=1)['Your estimated revenue (USD)'].reset_index(drop=True)
```

Scaling the data using Standard Scaler:

```
[ ] from sklearn.preprocessing import StandardScaler
    scaler=StandardScaler()
    scaler.fit(x_train)
    scaler.fit(x_test)
```

```
StandardScaler
StandardScaler()
```

1) Training the model using Random Forest & applying Grid Search CV for hyperparameter tuning:

> 1)Random forest is used over any other linear model since all the features were not linearly dependent on the target variable. Hence, using a linear model would not have been a suitable choice in this case.

> 2) Since the data is imbalanced using Random Forest is preferred over any other technique.

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

# Define the model
rf_model = RandomForestRegressor(random_state=50)

# Define the hyperparameter grid
param_grid = {
    'n_estimators':np.arange(100,330,15),
    'max_depth':np.arange(3,10,1),
    'max_features': ['sqrt', 'log2']
}
# Define the Grid Search object with R2 score as the scoring method
grid_search = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid,
    scoring='r2',
    cv=5,
    n_jobs=-1
)

# Fit the Grid Search object to the data
grid_search.fit(x_train,y_train)

# Retrieve the best R2 score and the best hyperparameters
best_hyperparams = grid_search.best_params_
```

```
[ ] best_hyperparams
```

```
{'max_depth': 8, 'max_features': 'sqrt', 'n_estimators': 220}
```

Tuning the model on best hyperparameters:

```
[ ] from sklearn.ensemble import RandomForestRegressor
    model=RandomForestRegressor(n_estimators=220,max_depth=8,max_features='sqrt',random_state=50)
    model.fit(x_train,y_train)
    y_pred=model.predict(x_test)
```

R2 Score:

```
[ ] from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
    r2_score(y_test,y_pred)
```

```
-0.593170782858037
```

Negative R2 Score signifies that the model does not capture a significant amount of varience. Hence, choosing another model will be an appropriate choice in this case.

Mean Absolute Error:

```
[ ] mean_absolute_error(y_test,y_pred)
```

```
34.60735338029757
```

Mean Squared Error:

```
[ ] mean_squared_error(y_test,y_pred)
```

```
1840.697109277877
```

R2 Score, mean absolute error & mean squared error are the evaluation metrics used to determine the model that best fits the data. It will help in selecting the appropriate regressor for the given dataset.

2) XG Boost Regressor: Using Grid Search CV for hyperparameter tuning.

> 1)XG Boost Regressor is selected since it combines all weak learners & sequentially learns from the errors made by the weak learners to build a robust model.

> 2) XG Boost Regressor performs the task of feature selection on its own which further makes it an appropriate model of choice in the regression problem given in this case.

> 3) XG Boost Regressor can handle both linear and non-linear relationships between the input features and the output variable. It uses a combination of decision trees and gradient boosting to capture complex relationships between the input features and the output variable.

```
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
param_grid = {
    'learning_rate': [0.1, 0.01],
    'n_estimators':np.arange(100,500,16),
    'max_depth': [3,4,5,6,7,8]
}
```

```
xgb=XGBRegressor()
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5, verbose=2)
grid_search.fit(x_train,y_train)
```

```
[CV] END ...learning_rate=0.1, max_depth=3, n_estimators=468; total time=   0.6s
[CV] END ...learning_rate=0.1, max_depth=3, n_estimators=468; total time=   0.5s
[CV] END ...learning_rate=0.1, max_depth=3, n_estimators=468; total time=   0.5s
[CV] END ...learning_rate=0.1, max_depth=3, n_estimators=484; total time=   0.5s
[CV] END ...learning_rate=0.1, max_depth=3, n_estimators=484; total time=   0.6s
[CV] END ...learning_rate=0.1, max_depth=3, n_estimators=484; total time=   0.6s
[CV] END ...learning_rate=0.1, max_depth=3, n_estimators=484; total time=   0.6s
[CV] END ...learning_rate=0.1, max_depth=3, n_estimators=484; total time=   2.8s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=100; total time=   0.1s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=100; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=100; total time=   0.1s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=100; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=100; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=116; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=116; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=116; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=116; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=116; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=132; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=132; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=132; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=132; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=132; total time=   0.2s
[CV] END ...learning_rate=0.1, max_depth=4, n_estimators=148; total time=   0.2s
```

```
best_estimator = grid_search.best_estimator_
test_score = best_estimator.score(x_train, y_train)
```

```
best_estimator
```

```
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=0.01, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=7, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

R2 Score:

```
test_score
```

```
0.7570357342885567
```

This is a good r2 score & the model captures a significant amount of varience. Hence, it will generalise well even on unseen data to give significant results.

Mean Squared Error:

```
[ ] mean_squared_error(y_test,y_pred)
```

```
1052.8185156341624
```

Mean Absolute Error:

```
[ ] mean_absolute_error(y_test,y_pred)
```

```
24.10280285810768
```

3) SVM:Using Grid Search CV for hyperparameter tuning.

Reasons for choosing SVM:

> 1)SVMs can effectively handle non-linear data by mapping input features to a high-dimensional space and finding a hyperplane that separates data points with the largest margin. SVMs can be effective on small to medium-sized datasets, as they are less prone to overfitting and can generalize well to new data points.

> 2) SVMs are suitable for regression tasks that require good generalization to new data, as they have good generalization capabilities.

> 3)SVMs are relatively robust to outliers in the data, which helps to improve the accuracy and robustness of the model.

```
[ ] from sklearn.svm import SVR
    svr=SVR()
    param_grid = {'C': [0.1,0.01,1,10],
                  'gamma': [0.1, 1, 10, 100],
                  'kernel': ['rbf','linear']}
    grid_search = GridSearchCV(svr, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
    grid_search.fit(x_train, y_train)
    y_pred = grid_search.best_estimator_.predict(x_test)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```

```
[ ] grid_search.best_estimator_
```

```
SVR
SVR(C=0.01, gamma=0.1)
```

R2 score:

```
[ ] svr=SVR('C':0.01,'gamma')
    r2_score(y_test,y_pred)
```

```
-2.4453407080619285
```

Negative R2 Score signifies that the model does not capture a significant amount of varience. Hence, choosing another model will be an appropriate choice in this case.

Mean Squared Error:

```
[ ] mean_squared_error(y_test,y_pred)
```

```
2276.791998161401
```

Mean Absolute Error:

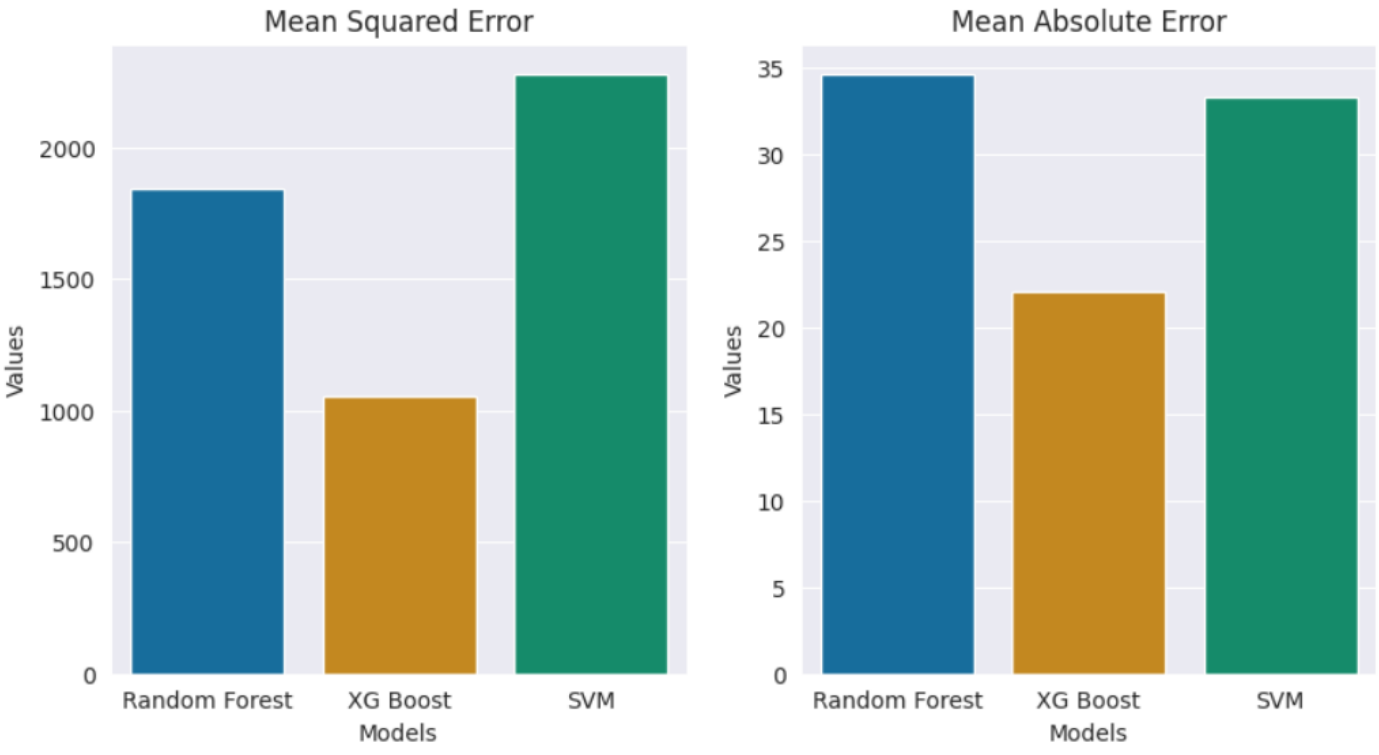
```
mean_absolute_error(y_test,y_pred)
```

```
33.2934
```

Conclusion:

> Comparison of MSE & MAE of all 3 models.

```
values1 = [1840, 1052, 2276.7919]
values2 = [34.6073, 22.1028, 33.2934]
labels1 = ["Random Forest", "XG Boost", "SVM"]
labels2 = ["Random Forest", "XG Boost", "SVM"]

sns.set_palette("colorblind")

# Create a grid of two barplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))

# Plot the first barplot
sns.barplot(x=labels1, y=values1, ax=axes[0])
axes[0].set_title("Mean Squared Error")
axes[0].set_xlabel("Models")
axes[0].set_ylabel("Values")

# Plot the second barplot
sns.barplot(x=labels2, y=values2, ax=axes[1])
axes[1].set_title("Mean Absolute Error")
axes[1].set_xlabel("Models")
axes[1].set_ylabel("Values")

# Show the plot
plt.show()
```



The above plot compared the evaluation parameters of all model & clearly indicates that XG Boost Regressor is the appropriate model to be used in this case. Since the mean squared error & mean absolute error of XG Boost Regressor is the least & the r2 score of XG Boost is maximum which indicates that it captures maximum varience & hence generalises unseen data better, it is the best suited model in this case

Model chosen: XG Boost Regressor

R2 score of regressor: 0.7570

MSE of regressor:1052.8185

MAE of regressor: 24.1028