Importing modified data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings(action="ignore")
plt.style.use(["seaborn-bright","dark_background"])
```

```
import os
os.getcwd()
```

```
'/content'
```

```
train = pd.read_csv("/modified_train.csv")
test = pd.read_csv("/modified_test.csv")
```

```
X = train.drop(columns=["Item_Outlet_Sales"])
y = train["Item_Outlet_Sales"]
```

```
from sklearn.model_selection import train_test_split
x_train, x_valid, y_train, y_valid = train_test_split(X,y,test_size=0.2,random_state=101)
```

Model training and evaluation.

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
import time
```

```
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,ExtraTreesRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
```

```
models = []
models.append(("LinearRegressor",LinearRegression()))
models.append(("Lasso",Lasso()))
models.append(("Ridge",Ridge()))
models.append(("DecisionTree",DecisionTreeRegressor(criterion="mse")))
models.append(("RandomForestRegressor",RandomForestRegressor(criterion="mse")))
models.append(("ExtraTreeRegressor",ExtraTreesRegressor()))
models.append(("AdaBoostRegressor",AdaBoostRegressor()))
models.append(("GradientBoosingRegressor",GradientBoostingRegressor()))
models.append(("SVR",SVR()))
models.append(("KNeighborsRegressor",KNeighborsRegressor()))
```

```
names = []
score = []
rmse = []
mse = []
mae = []
r2 = []
timing = []
```

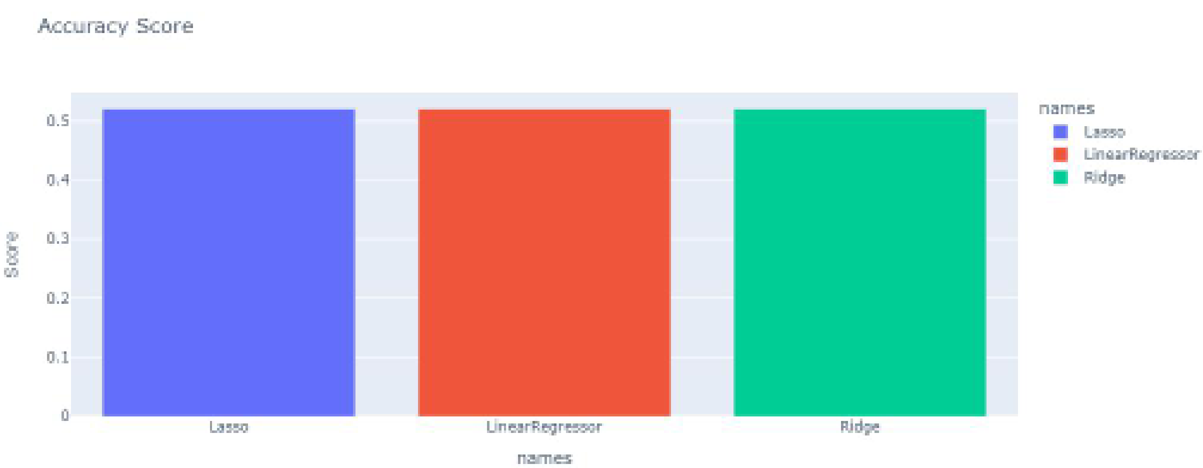
```
for name,model in models:
    model = model
    beg = time.time()
    model.fit(x_train,y_train)
    pred = model.predict(x_valid)
    end = time.time()
    print("Model = {}".format(name))
    print("Score = {}, RMSE = {}, MSE = {}, MAE = {}, R2 = {},Time = {}\n".format(round(model.score(x_valid,y_valid),4),
                                                round(np.sqrt(mean_squared_error(pred,y_valid)),4),
                                                round(mean_squared_error(pred,y_valid),4),
                                                round(mean_absolute_error(pred,y_valid),4),
```

```
                                                    round(r2_score(y_valid,pred),4),(end-beg)))
    names.append(name)
    score.append(round(model.score(x_valid,y_valid),4))
    rmse.append(round(np.sqrt(mean_squared_error(pred,y_valid)),4))
    mse.append(round(mean_squared_error(pred,y_valid),4))
    mae.append(round(mean_absolute_error(pred,y_valid),4))
    r2.append(round(r2_score(pred,y_valid),4))
    timing.append(end-beg)
```

```
df = pd.DataFrame()
df["names"] = names
df["Score"] = score
df["time"] = timing
df["rmse"] = rmse
df["mse"] = mse
df["mae"] = mae
df["r2"] = r2
```

```
df = df.sort_values(by="Score",ascending=False)
fig = px.bar(df,"names","Score",color="names",title="Accuracy Score")
fig.show()
```

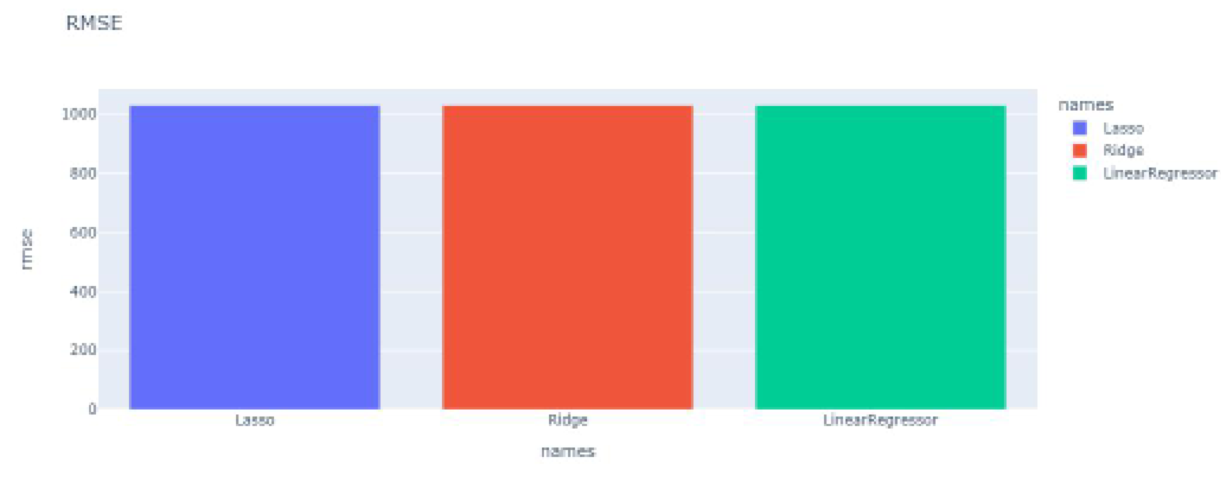
Accuracy Score



+ Code + Text

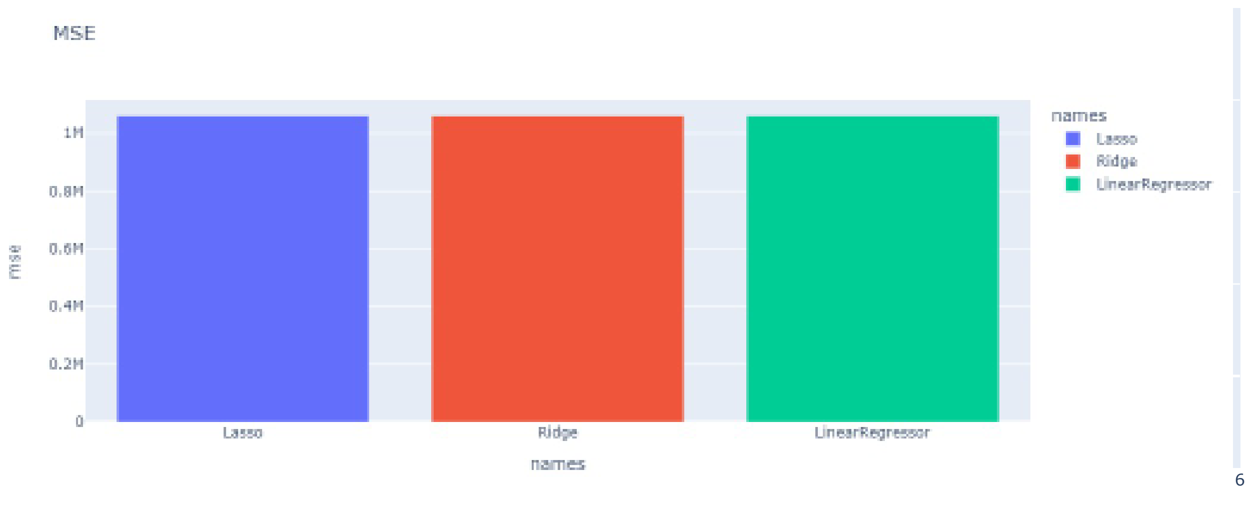
```
df = df.sort_values(by="rmse",ascending=True)
fig = px.bar(df,"names","rmse",color="names",title="RMSE")
fig.show()
```

RMSE




```
df = df.sort_values(by="mse",ascending=True)
fig = px.bar(df,"names","mse",color="names",title="MSE")
fig.show()
```

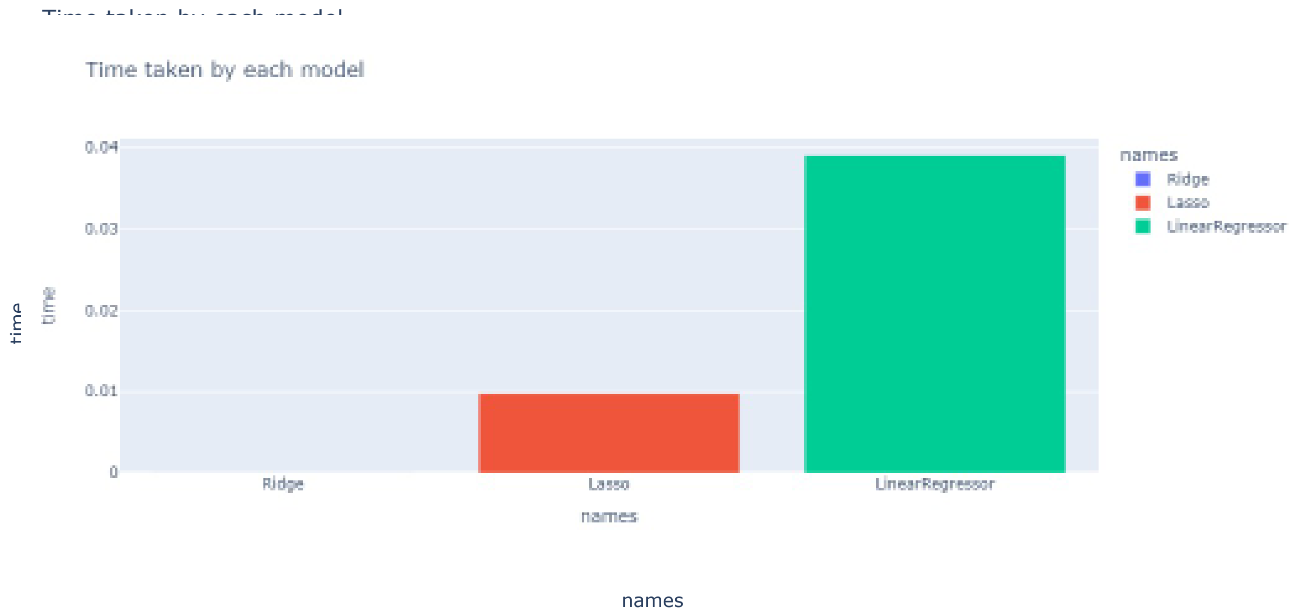
MSE




```
df = df.sort_values(by="mae",ascending=True)
fig = px.bar(df,"names","mae",color="names",title="MAE")
fig.show()
```

```
df = df.sort_values(by="r2",ascending=True)
fig = px.bar(df,"names","r2",color="names",title="R2 Score")
fig.show()
```


```
df = df.sort_values(by="time",ascending=True)
fig = px.bar(df,"names","time",color="names",title="Time taken by each model")
fig.show()
```




```
# On the basis of all above metrices and time taken by the model we choose the GradientBoostingRegressor.
```

```
#Below cell will take much time to complete. Have patience.
```

Hyperparameter tuning.

```
from sklearn import model_selection
classifer = GradientBoostingRegressor()
dt_grid = {'loss':['ls', 'lad', 'huber'],
           'learning_rate':[0.01,0.05,0.1,0.5],
          'n_estimators' : [100,200],
          'max_depth':[3,5,8,10]}
grid_classifier = model_selection.GridSearchCV(classifer, dt_grid, cv=10, refit=True,
                                               return_train_score=True)
grid_classifier.fit(X, y)
results = grid_classifier.cv_results_
print(results.get('params'))
print(results.get('mean_test_score'))
print(results.get('mean_train_score'))
print(grid_classifier.best_params_)
print(grid_classifier.best_score_)
final_model = grid_classifier.best_estimator_
```

```
[{'learning_rate': 0.01, 'loss': 'ls', 'max_depth': 3, 'n_estimators': 100}, {'learning_rate': 0.01, 'loss': 'ls', 'max_depth': 3, 'n_estimators': 200
[0.45666354 0.5383241  0.48639658 0.55214666 0.48356294 0.54289183
 0.47475018 0.52884411 0.41172121 0.50903553 0.4563813  0.53846537
 0.4559354  0.53120754 0.45145131 0.52270371 0.44465863 0.53200664
 0.47660263 0.54837358 0.47345107 0.53773397 0.46421036 0.5252806
 0.56219903 0.55898499 0.5578043  0.54757802 0.54253485 0.5230378
 0.51707634 0.49204565 0.55527959 0.55695709 0.55687108 0.55511645
 0.54269681 0.54141506 0.53111017 0.53135754 0.56080537 0.55835633
 0.55615383 0.54802692 0.54092345 0.52414699 0.5202874  0.49699505
 0.55875926 0.55197103 0.54627866 0.5321496  0.51876295 0.49125825
 0.48673411 0.46290662 0.55670994 0.55701126 0.55499688 0.55357483
 0.54300196 0.53971384 0.53260159 0.52492466 0.55871537 0.5520153
 0.54773068 0.53261227 0.51619728 0.4957597  0.49439778 0.46594476
 0.5155621  0.48658162 0.45113661 0.39913738 0.36877039 0.34593855
 0.34136894 0.33905061 0.55212045 0.5522085  0.54296472 0.54205124
 0.5123446  0.50850891 0.47423237 0.47005396 0.51607627 0.4931469
 0.45444863 0.40817854 0.3685745  0.33761779 0.35153939 0.3449091 ]
[0.4614503  0.54480375 0.49479414 0.56771154 0.53587114 0.62805944
 0.58708333 0.70057    0.41432976 0.5129928  0.46278081 0.54835934
 0.4856318  0.57953852 0.50915615 0.61503382 0.44928354 0.53806567
 0.48530486 0.56373609 0.521811   0.6167868  0.56385221 0.67718955
 0.57700238 0.58994745 0.60493998 0.64472983 0.70162853 0.78821659
 0.79290549 0.88414095 0.56235384 0.57014873 0.57993772 0.58838613
 0.62471353 0.63238344 0.67046719 0.67979937 0.57548673 0.58839026
 0.60355567 0.63842554 0.6891488  0.77083813 0.7727666  0.86364983
 0.59098229 0.61509492 0.64633342 0.71000828 0.79003584 0.87945228
 0.88946099 0.95312369 0.57025086 0.57354789 0.58822297 0.59045152
 0.63439349 0.63710729 0.68094054 0.69299451 0.58882959 0.61032526
 0.64031666 0.69784656 0.77657243 0.86146278 0.86549791 0.93586116
 0.66995309 0.732847   0.82114121 0.9082088  0.97316366 0.99699947
 0.99690316 0.99993363 0.58002685 0.57964509 0.60411933 0.60087394
 0.65668795 0.66120262 0.7142575  0.72134682 0.66264662 0.72045787
 0.80111138 0.8828194  0.95322973 0.98867128 0.98944244 0.99869967]
{'learning_rate': 0.05, 'loss': 'ls', 'max_depth': 3, 'n_estimators': 100}
0.5621990292342698
```

```
pred  = final_model.predict(x_valid)
mean_absolute_error(pred,y_valid)
```

Saving best model for deployment.

```
import pickle
```

```
pickle_out = open("sales_flask.pkl","wb")
pickle.dump(final_model,pickle_out)
```

```
0.5749856087550711
```

Loading model.

```
loaded_model = pickle.load(open("sales_flask.pkl","rb"))
pred = loaded_model.predict(test)
```

```
pred
```

```
array([1662.3091974 , 1381.05716057,  643.17042028, ..., 1902.46738382,
       3384.9306051 , 1326.95857518])
```