Restaurant Revenue Prediction

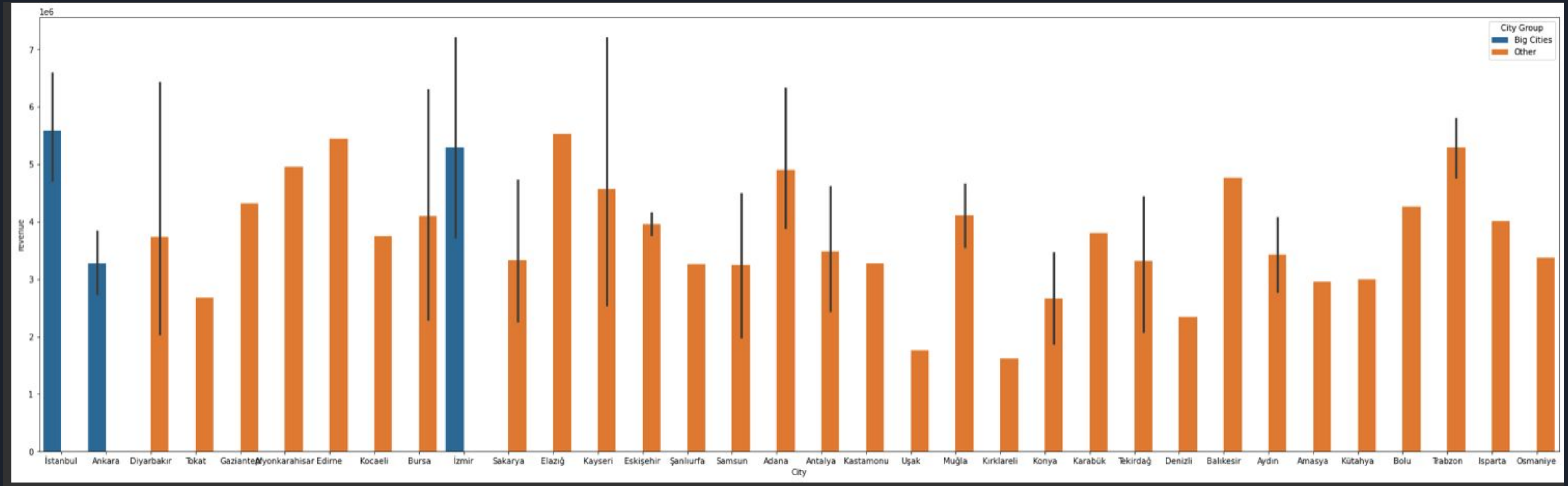
Odeia Busheri
Simon Bruno
Eddie Rudoy

Reminder: Dataset

- The train set consist of 137 rows (samples) and 43 columns which represent the features.
- **Id** : Restaurant id.
- **Open Date** : opening date for a restaurant
- **City** : City that the restaurant is in.
- **City Group**: Type of the city. Big cities, or Other.
- **Type**: Type of the restaurant. FC: Food Court, IL: Inline, DT: Drive Thru, MB: Mobile
- **P1, P2 - P37**: There are three categories of these obfuscated data.
Rank of **demographic data-** population in any given area, age and gender distribution, development scales.
Real estate data- mainly relate to the m2 of the location, front facade of the location, car park availability.
Commercial data- mainly include the existence of points of interest.
- **Revenue**: The revenue column indicates a (*transformed) revenue of the restaurant in a given year and is the target of predictive analysis. Transformed indicates that these values don't mean real dollar value.

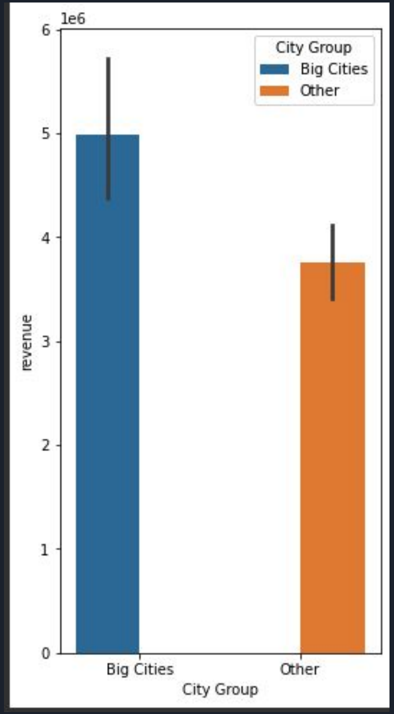
Categorial Features

City vs Revenue



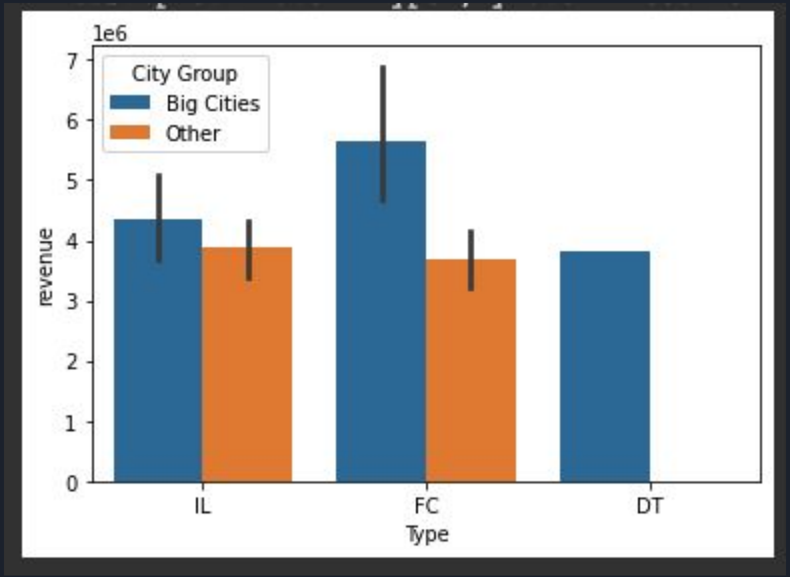
Categorial Features

City Group vs Revenue



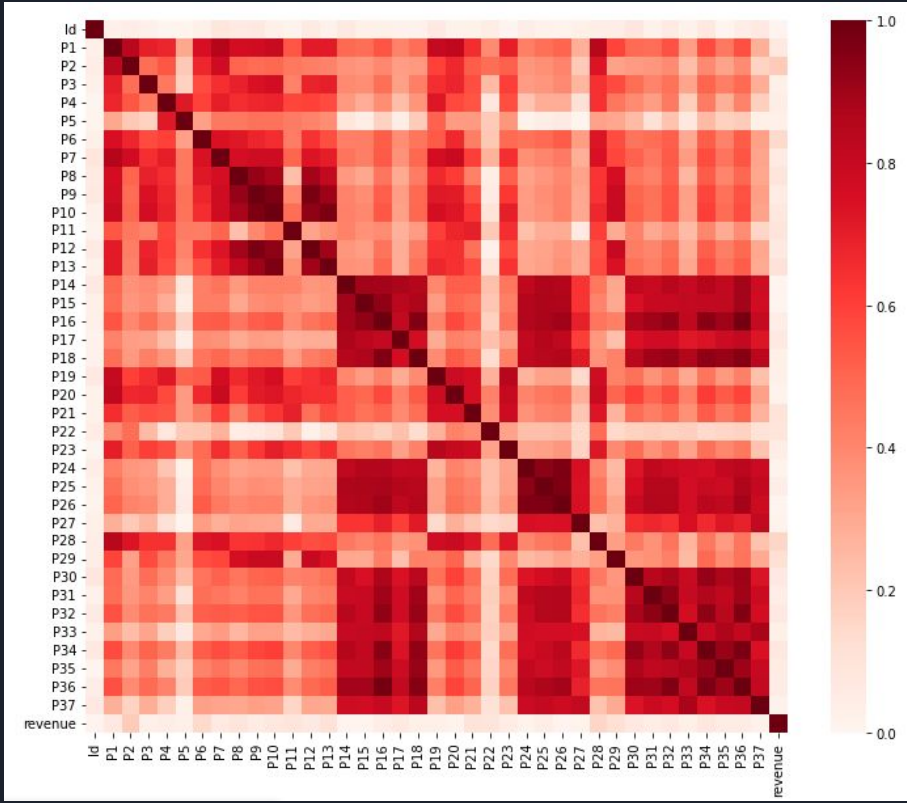
Categorial Features

Type (City Group) vs Revenue



Heatmap - P variables

P-values heatmap

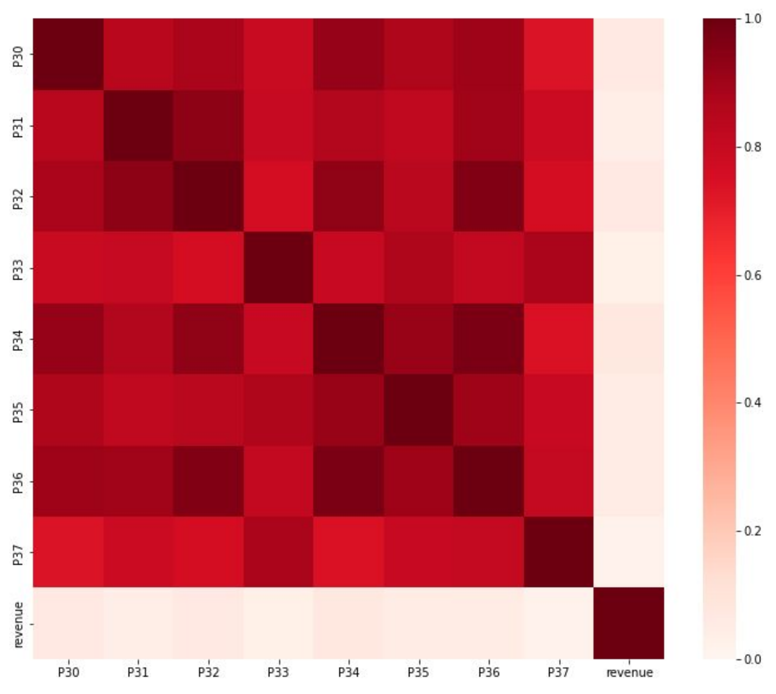


As we can see, there are groups of highly correlated features amongst each other, let's take a closer look

Heatmap - P variables

P(30-37)-values heatmap:

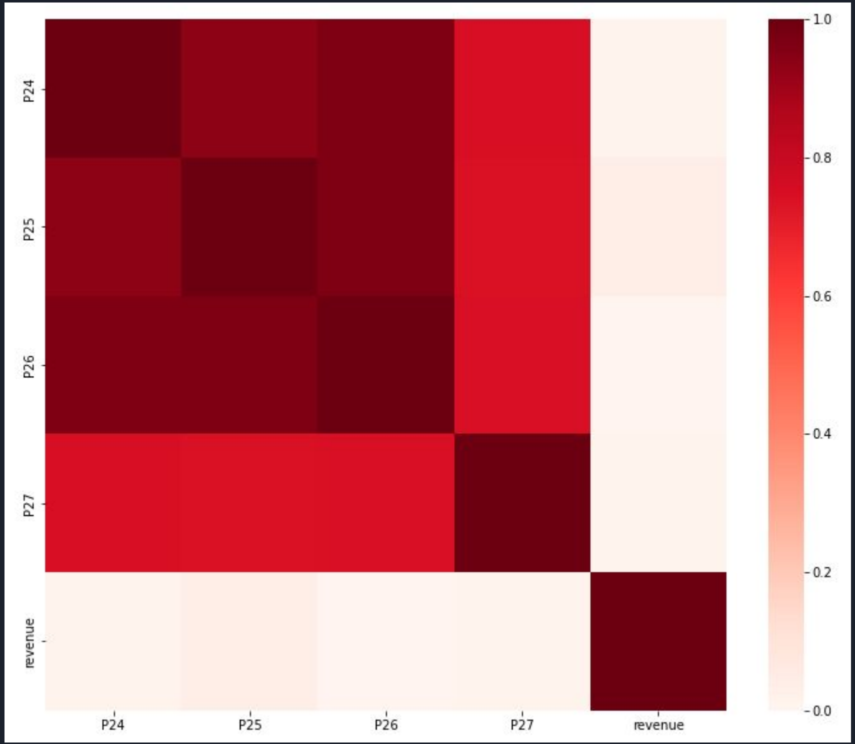
We can see we have a very high correlation among the P values themselves, and a low correlation to the revenue, we'll lated remove all but one



Heatmap - P variables

P(24-27)-values heatmap:

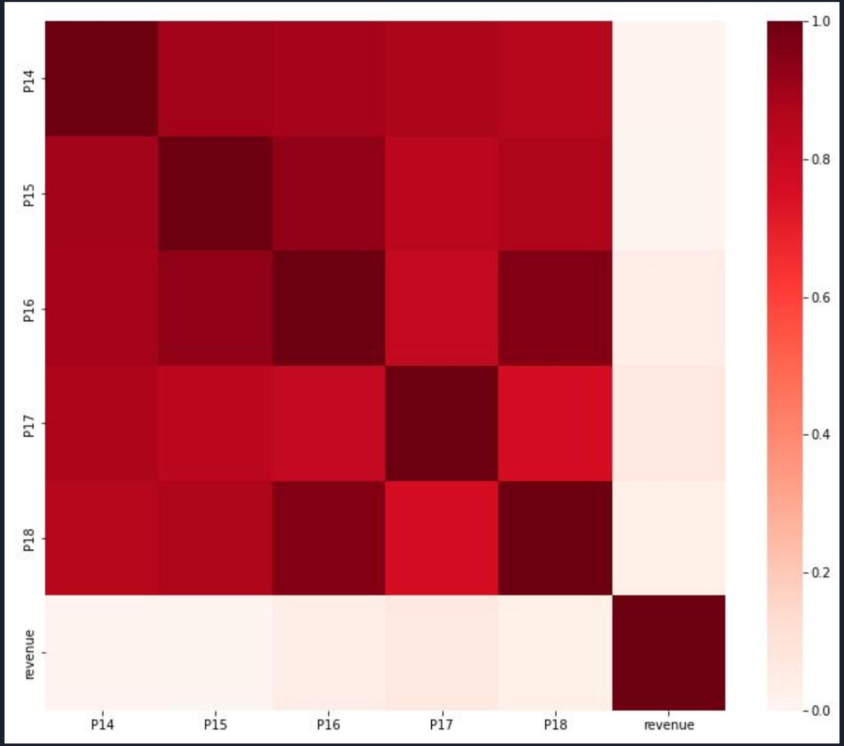
Same thing here



Categorial Features

P(14-18)-values heatmap:

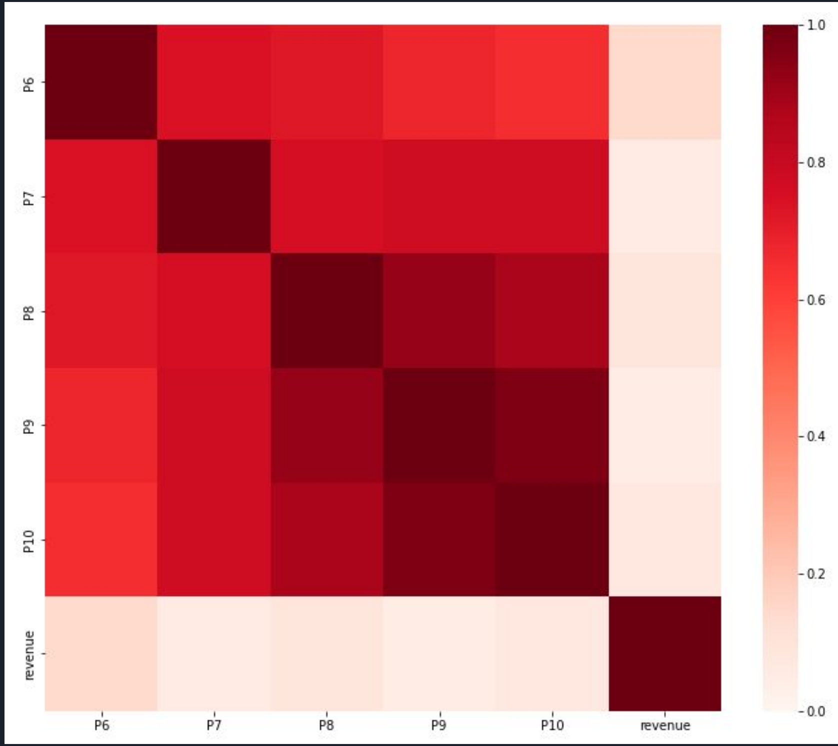
Also here



Heatmap - P variables

P(30-37)-values heatmap:

And lastly here



P variables - correlations

For each of the above groups of P-values, we'll examine the highest correlation to the revenue, and drop all but that one.

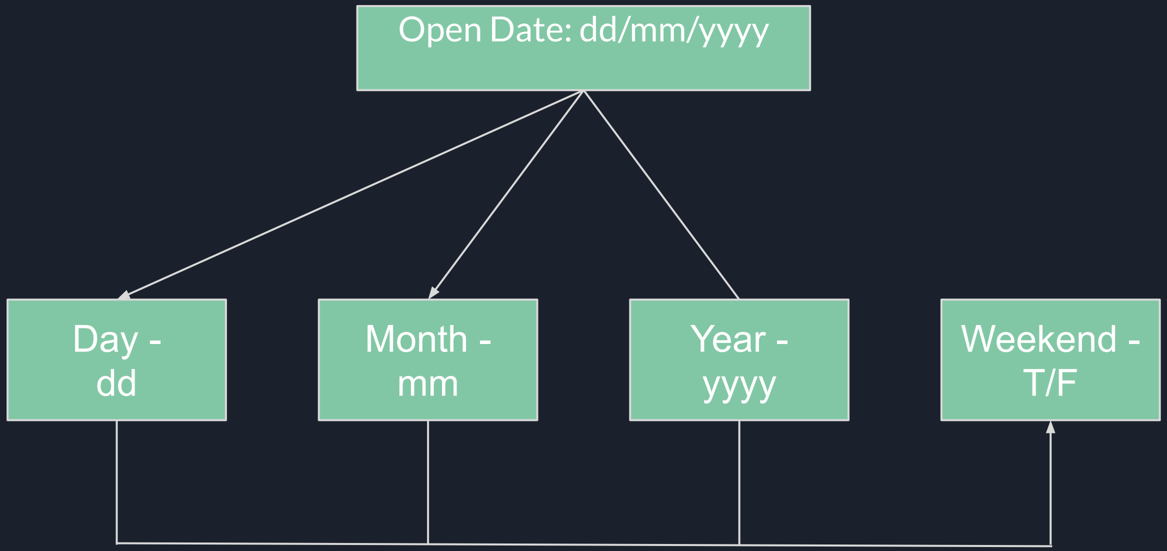
```
P14     0.006441
P15     0.000742
P16     0.037997
P17     0.067137
P18     0.034537
Name: revenue, dtype: float64
0.0671369027405342
```

```
P24     0.014222
P25     0.036365
P26     0.007650
P27     0.013680
Name: revenue, dtype: float64
0.03636464638628595
```

```
P6      0.139094
P7      0.051165
P8      0.084215
P9      0.050352
P10     0.073220
Name: revenue, dtype: float64
0.13909423890893735
```

```
P30     0.066203
P31     0.040418
P32     0.065857
P33     0.032426
P34     0.072343
P35     0.050156
P36     0.050534
P37     0.019051
Name: revenue, dtype: float64
0.07234280699800917
```

Feature Engineering

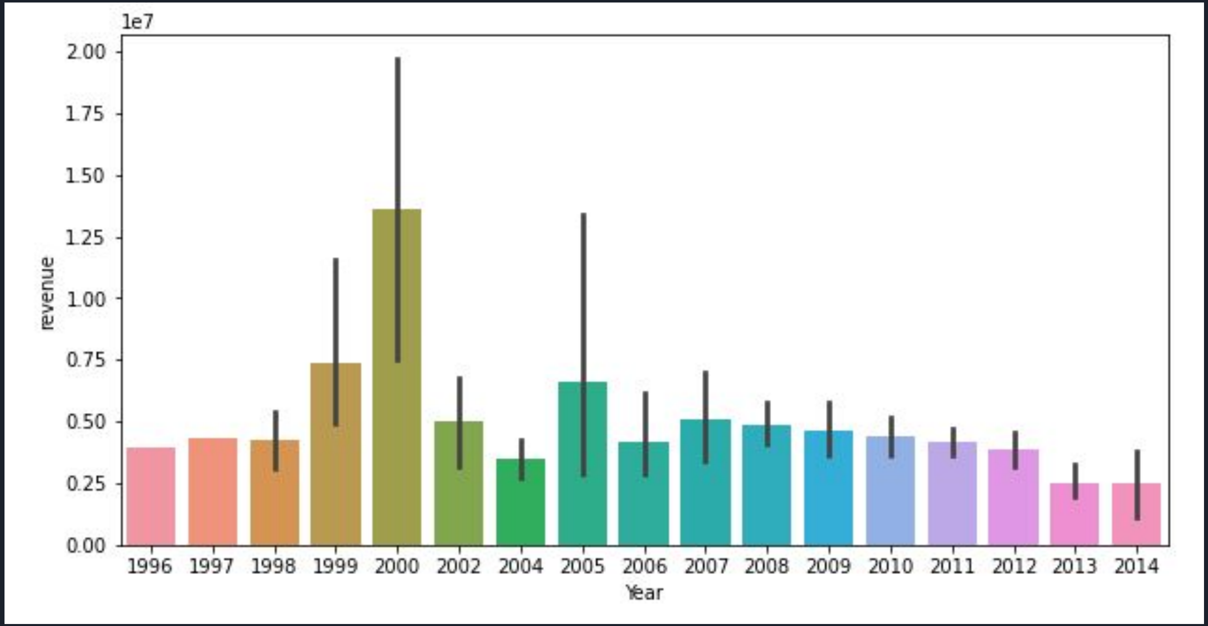


Calculate weekend from date

Feature Engineering

Let’s examine how the newly extracted fields correlate to the revenue!

We’ll start with the year:

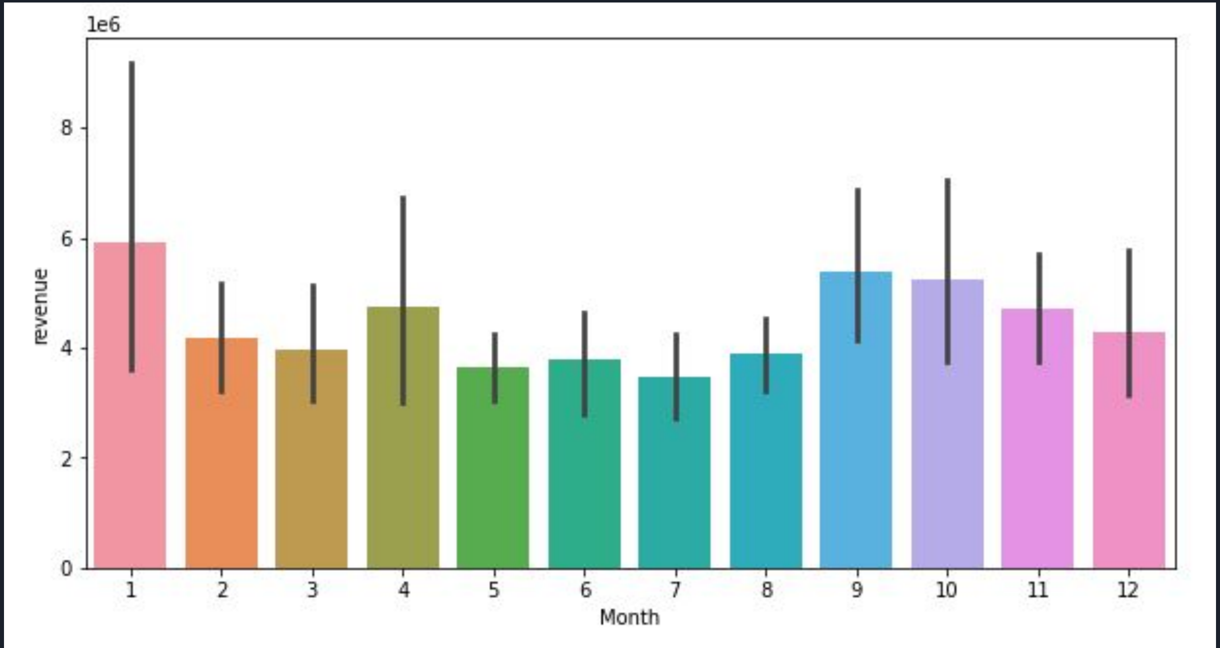


It seems that other than the start of the millenia the year didn’t really correlate to the revenue

Feature Engineering

Let's examine how the newly extracted fields correlate to the revenue!

Now, let's look at the month:

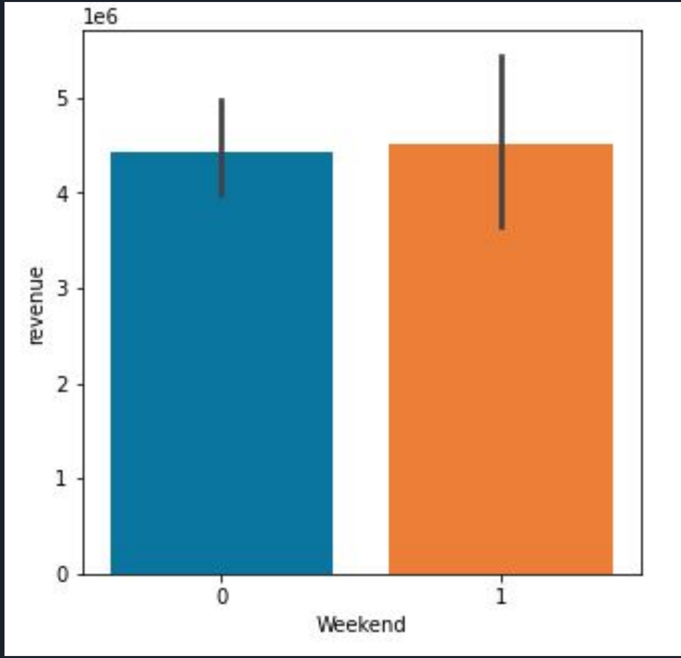


We see somewhat of a trend for colder months, but nothing that stands out significantly

Feature Engineering

Let’s examine how the newly extracted fields correlate to the revenue!

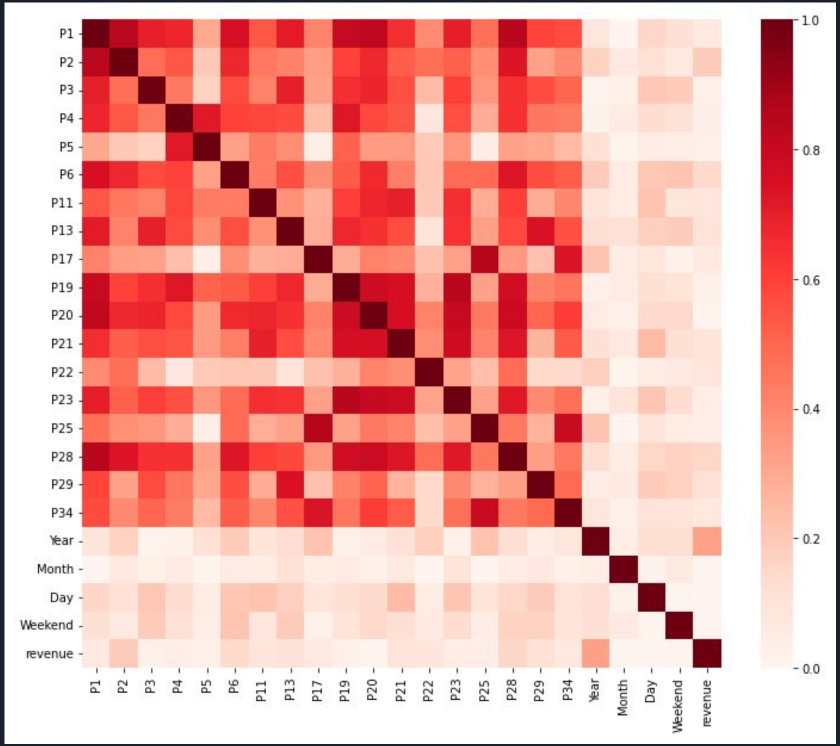
Last but not least is the weekend:



Unfortunately no large differences here as well. Some more deviation for the weekends but nothing more.

Feature Engineering

Let's take one last gander at our resulting correlation heatmap with the new date related features and the dropped P-values



Libraries & Imports

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import math
import seaborn as sns
import matplotlib.pyplot as plt
import scipy
import plotly.express as px
import plotly.graph_objects as go

# sklearn imports
import sklearn
from sklearn import metrics
from sklearn import datasets
from sklearn import pipeline
from sklearn import model_selection
from sklearn import metrics
from sklearn import pipeline
from sklearn import linear_model
from sklearn import preprocessing
from sklearn import model_selection
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import LeavePOut
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

Read Data

```
#Read data

data_path = '/content/train.csv.zip'
test_path = '/content/test.csv.zip'

restaurant_df = pd.read_csv(data_path)
test_df = pd.read_csv(test_path)

display(restaurant_df)

display(test_df)
```

Data Exploration

Check for empty values in the train set

```python
print("Check if there are columns with empty values")
display(restaurant_df.isna().any())

print(f'There are {len(np.where(restaurant_df.isnull())[0])} empty values in the dataframe')

# count empty values in each column
def count_empty_values_in_each_column(df: pd.DataFrame):
  print('empty values')
  print('------------\n')

  for col in df.columns:
    print(f"{col}: {df[col].isna().sum()}")

count_empty_values_in_each_column(restaurant_df)
```

```
Check if there are columns with empty value
Id            False
Open Date     False
City          False
City Group    False
Type          False
P1            False
P2            False
P3            False
P4            False
P5            False
P6            False
P7            False
P8            False
P9            False
P10           False
P11           False
P12           False
P13           False
P14           False
P15           False
P16           False
P17           False
P18           False
P19           False
P20           False
P21           False
P22           False
P23           False
P24           False
P25           False
P26           False
P27           False
P28           False
P29           False
P30           False
P31           False
P32           False
P33           False
P34           False
P35           False
P36           False
P37           False
revenue       False
dtype: bool
There are 0 empty values in the dataframe
empty values
------------
```

```
There are 0 empty values in the dataframe
empty values
------------
```

Divide the data to features and target

```python
# divide the data to features and target
t = restaurant_df_cp['revenue'].copy()
X = restaurant_df_cp.drop(['revenue'], axis=1)
print('t')
display(t)
print()
print('X')
display(X)
```

```
t
0      5653753.0
1      6923131.0
2      2055379.0
3      2675511.0
4      4316715.0
         ...
132    5787594.0
133    9262754.0
134    2544857.0
135    7217634.0
136    6363241.0
Name: revenue, Length: 137, dtype: float64

X
```

	City	City Group	Type	P1	P2	P3	P4	P5	P6	P11	P13	P17	P19	P20	P21	P22	P23	P25	P28	P29	P34	Year	Month	Day	Weekend
0	İstanbul	Big Cities	IL	4	5.0	4.0	4.0	2	2	3	5.0	2	5	4	1	3	3	1	2.0	3.0	5	1999	7	17	1
1	Ankara	Big Cities	FC	4	5.0	4.0	4.0	1	2	1	5.0	0	3	2	1	3	2	0	3.0	3.0	0	2008	2	14	0
2	Diyarbakır	Other	IL	2	4.0	2.0	5.0	2	3	2	5.0	0	1	1	1	1	1	0	1.0	3.0	0	2013	3	9	1
3	Tokat	Other	IL	6	4.5	6.0	6.0	4	4	8	7.5	3	20	12	6	1	10	2	2.5	7.5	18	2012	2	2	0
4	Gaziantep	Other	IL	3	4.0	3.0	4.0	2	2	2	5.0	1	2	2	1	2	1	3	1.0	3.0	3	2009	5	9	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
132	Trabzon	Other	FC	2	3.0	3.0	5.0	4	2	4	4.0	0	4	3	2	1	1	0	2.0	3.0	0	2008	6	25	0
133	İzmir	Big Cities	FC	4	5.0	4.0	4.0	2	3	5	5.0	0	3	2	2	1	1	0	3.0	3.0	0	2006	10	12	0
134	Kayseri	Other	FC	3	4.0	4.0	4.0	2	3	1	5.0	0	2	3	1	2	2	0	2.0	3.0	0	2006	7	8	1
135	İstanbul	Big Cities	FC	4	5.0	4.0	5.0	2	2	2	5.0	0	1	1	1	1	1	0	3.0	3.0	0	2010	10	29	0
136	İstanbul	Big Cities	FC	4	5.0	3.0	5.0	2	2	4	5.0	0	2	1	1	1	1	0	3.0	3.0	0	2009	9	1	0

137 rows × 25 columns

Cross Validation Score Function

```python
# calculate score and loss from cv (KFold) and display graphs
def get_cv_score_and_loss(X, t, model, k=None, p=None, show_score_loss_graphs=False, use_pbar=True):
    scores_losses_df = pd.DataFrame(columns=['fold_id', 'split', 'score', 'loss'])

    if k is not None:
        cv = KFold(n_splits=k, shuffle=True, random_state=1)

    else:
        raise ValueError('you need to specify k or p in order for the cv to work')

    for i, (train_ids, val_ids) in enumerate(cv.split(X)):
        X_train = X.loc[train_ids]
        t_train = t.loc[train_ids]
        X_val = X.loc[val_ids]
        t_val = t.loc[val_ids]

        model.fit(X_train, t_train)

        y_train = model.predict(X_train)
        y_val = model.predict(X_val)
        scores_losses_df.loc[len(scores_losses_df)] = [i, 'train', model.score(X_train, t_train), mean_squared_error(t_train, y_train, squared=False)]
        scores_losses_df.loc[len(scores_losses_df)] = [i, 'val', model.score(X_val, t_val), mean_squared_error(t_val, y_val, squared=False)]


    val_scores_losses_df = scores_losses_df[scores_losses_df['split']=='val']
    train_scores_losses_df = scores_losses_df[scores_losses_df['split']=='train']

    mean_val_score = val_scores_losses_df['score'].mean()
    mean_val_loss = val_scores_losses_df['loss'].mean()
    mean_train_score = train_scores_losses_df['score'].mean()
    mean_train_loss = train_scores_losses_df['loss'].mean()

    if show_score_loss_graphs:
        fig = px.line(scores_losses_df, x='fold_id', y='score', color='split', title=f'Mean Val Score: {mean_val_score:.2f}, Mean Train Score: {mean_train_score:.2f}')
        fig.show()
        fig = px.line(scores_losses_df, x='fold_id', y='loss', color='split', title=f'Mean Val Loss: {mean_val_loss:.2f}, Mean Train Loss: {mean_train_loss:.2f}')
        fig.show()

    return mean_val_score, mean_val_loss, mean_train_score, mean_train_loss
```

Divide the cols for encoding

```
[102] numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns
      categorical_cols = X.select_dtypes(include=['object', 'bool']).columns
      all_cols = np.array(X.columns)
```

```
from sklearn.compose import ColumnTransformer

ct = ColumnTransformer([
    ("encoding", OneHotEncoder(sparse=False, handle_unknown='ignore'), categorical_cols),
    ("standard", StandardScaler(), numerical_cols)])
```

One-Hot Encoding:
Transform the categorical data into few binary columns. Translate each category into a column with 0 and 1 values.

Modeling

Linear Model- SGDRegressor

XGBoost

AdaBoost

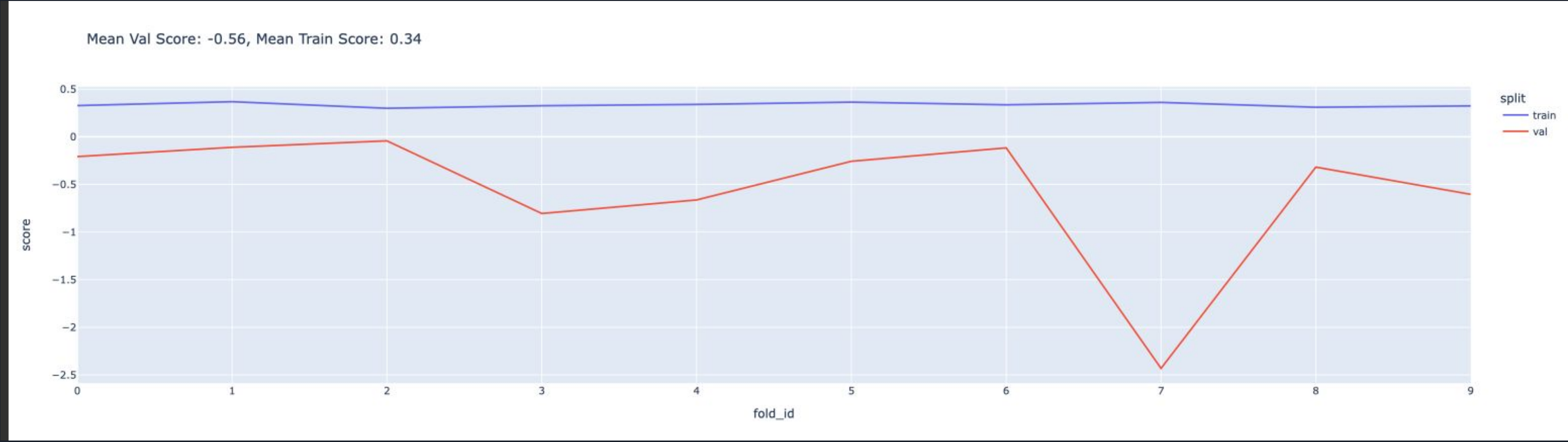
Random Forest

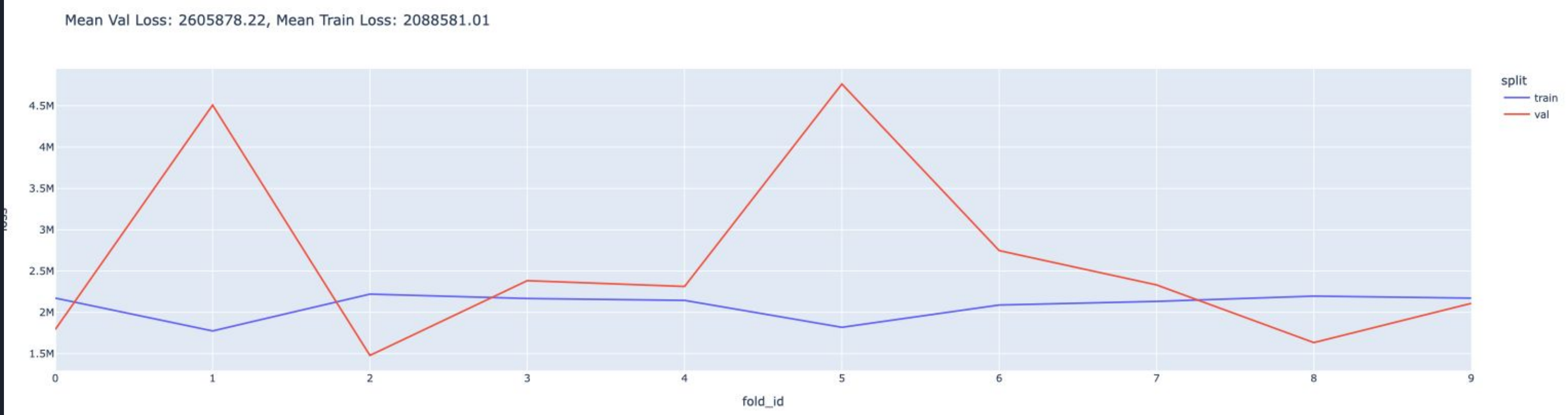
SGDRegressor

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate)

```
model_pipe = make_pipeline(ct, SGDRegressor(random_state=1))
val_score, val_loss, train_score, train_loss = get_cv_score_and_loss(X, t, model_pipe, k=10, show_score_loss_graphs=True)
print(f'mean cv val score: {val_score:.2f}\nmean cv val loss {val_loss:.2f}')
print(f'mean cv val score: {train_score:.2f}\nmean cv val loss {train_loss:.2f}')
```

Mean cv val score: -0.56
Mean cv val loss: 2605878.22
Mean cv train score: 0.34
Mean cv val loss: 2088581.01


Mean Val Score: -0.56, Mean Train Score: 0.34
score
fold_id
split
train
val


Mean Val Loss: 2605878.22, Mean Train Loss: 2088581.01
fold_id
split
train
val

SGDRegressor Train Result

```
[35] SDG_reg_original = pipeline.make_pipeline(ct, SGDRegressor(random_state=1)).fit(X, t)

     y_train_sdg = SDG_reg_original.predict(X)
     print('Accuracy score on train', SDG_reg_original.score(X, t))
     print('RMSE score on train',(mean_squared_error(t, y_train_sdg, squared=False)))
```

```
Accuracy score on train 0.31139232096021396
RMSE score on train 2129869.676142312
```

Hyper Parameter Search - using GridSearchCV

Most of our models have a lot of parameters that can be adjusted.

Each parameter value can make our model better (or worse).

We want to be able to find the best hyperparameters for our models.

Grid Search: When we want to check every parameter possible, we will use Grid Search.

We will try all combinations of parameters and find the best one, that gives us the best score.

SGDRegressor Hyper Parameters

Penalty, alpha

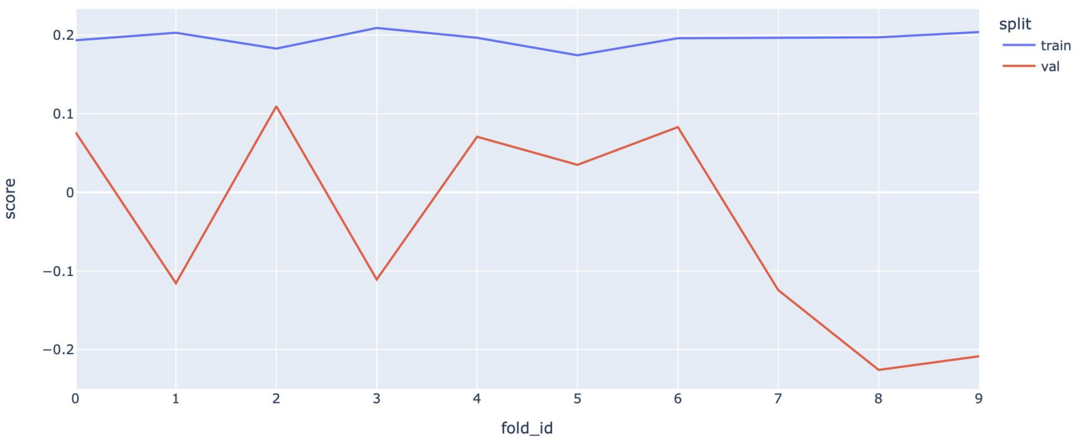
```
[83] feature_arr = ct.fit_transform(X, t)
     feature_labels = ct.get_feature_names_out(all_cols.tolist())
     X_encoded = pd.DataFrame(feature_arr, columns=feature_labels)

     hyper_parameters = {'penalty': ('l2', 'l1', 'elasticnet'), 'alpha':[0.0001, 0.001, 0.01, 0.1,1]}

     gs_model = GridSearchCV(SGDRegressor(random_state=1), hyper_parameters).fit(X_encoded, t)
     print('Accuracy score for regression:')
     print('gs_model', gs_model.best_score_)
     print('best params', gs_model.best_params_)
```

```
Accuracy score for regression:
gs_model 0.009642228268009246
best params {'alpha': 1, 'penalty': 'l2'}
```

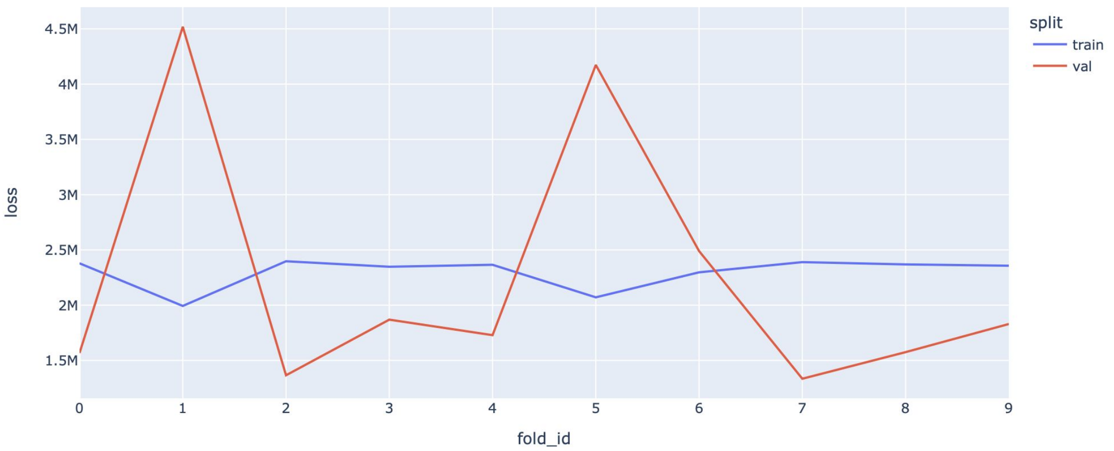
```
[82] y_train_hyper = gs_model.predict(X_encoded)

     print('Accuracy score on train', gs_model.score(X_encoded, t))
     print('RMSE score on train',(mean_squared_error(t, y_train_hyper, squared=False)))
```

```
Accuracy score on train 0.18460765025037884
RMSE score on train 2317663.684533361
```

Mean Val Score: -0.04, Mean Train Score: 0.20



Mean Val Loss: 2245343.15, Mean Train Loss: 2296667.47



XGBoost - XGBRegressor

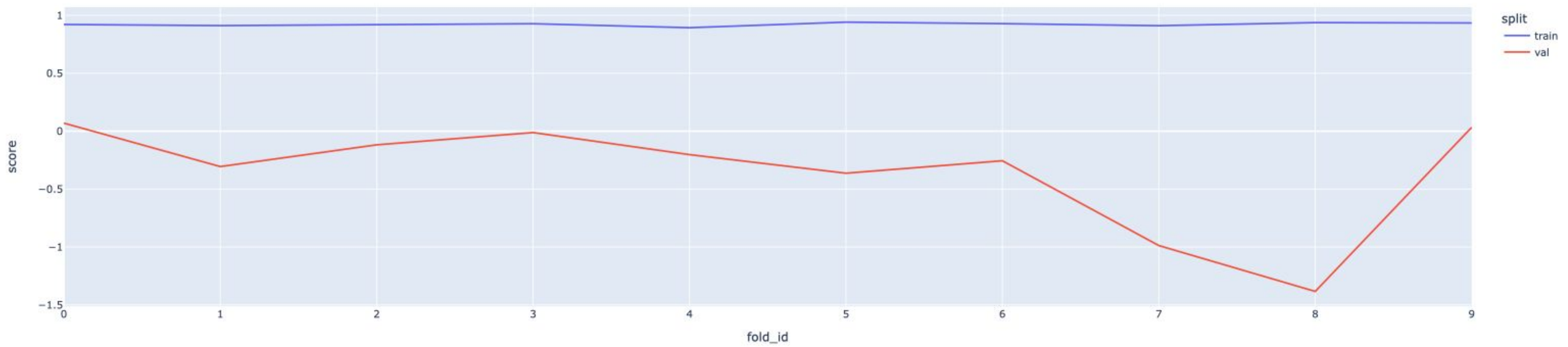
XGBoost is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**.

```
from xgboost import XGBRegressor
```

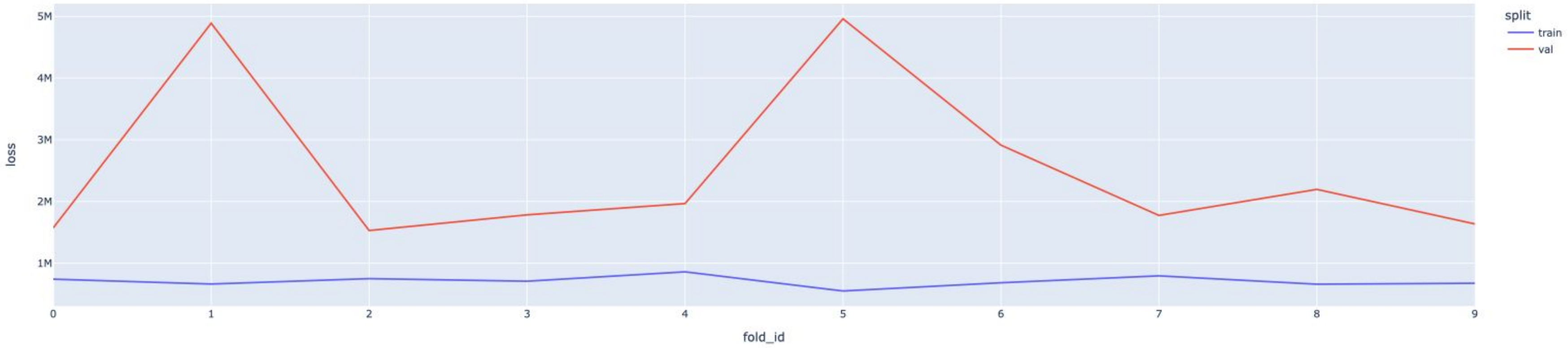
```
[32] XGB_model = pipeline.make_pipeline(ct, XGBRegressor(objective='reg:squarederror'))
     val_score, val_loss, train_score, train_loss = get_cv_score_and_loss(X, t, XGB_model, k=10, show_score_loss_graphs=True)
     print(f'mean cv val score: {val_score:.2f}\nmean cv val loss {val_loss:.2f}')
     print(f'mean cv val score: {train_score:.2f}\nmean cv val loss {train_loss:.2f}')
```

Mean cv val score: -0.35
Mean cv val loss: 2522287.70
Mean cv train score: 0.92
Mean cv val loss: 709300.20

Mean Val Score: -0.35, Mean Train Score: 0.92



Mean Val Loss: 2522287.70, Mean Train Loss: 709300.20

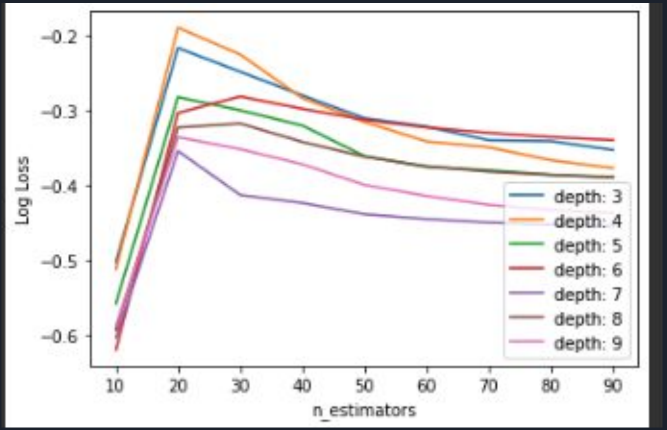


XGBoost Hyperparameters

N_estimators, max_depth

```
param_dist = {'n_estimators': list(range(10,100,10)),
              'max_depth': range(3,10),
             }

rs_model = GridSearchCV(XGBRegressor(objective='reg:squarederror'), param_dist).fit(X_encoded, t)
print('Accuracy score for regression:')
print('rs_model', rs_model.best_score_)
print('best params', rs_model.best_params_)
```

```
Accuracy score for regression:
rs_model -0.19011091740377947
best params {'max_depth': 4, 'n_estimators': 20}
```

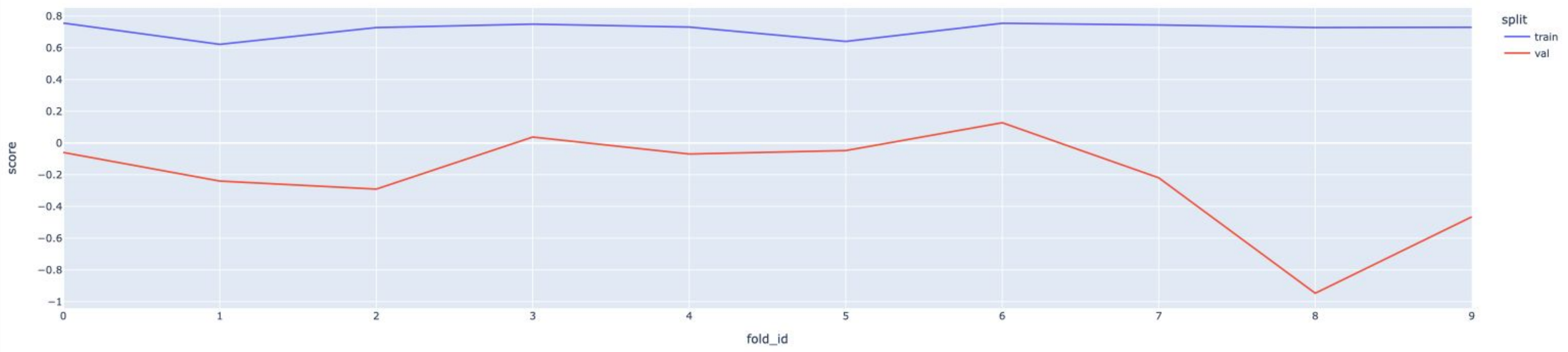


XGBoost Training Score With Hyperparameters

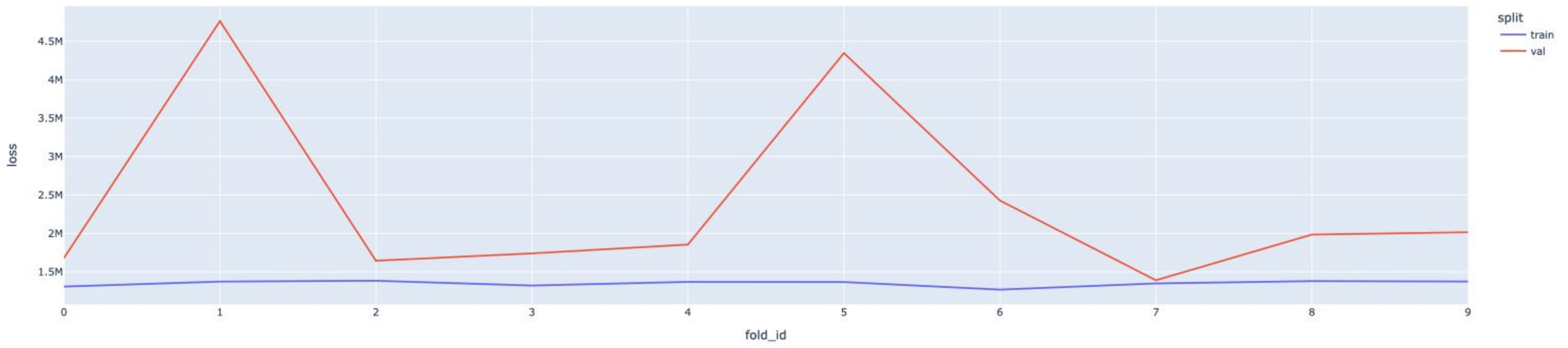
```
rs_model_pipe  = pipeline.make_pipeline(ct, XGBRegressor(objective='reg:squarederror', max_depth=4, n_estimators=20)).fit(X, t)
print('Accuracy score for regression:')
y_rs_pipe = rs_model_pipe.predict(X)
print('Accuracy score on train', rs_model_pipe.score(X, t))
print('RMSE score on train',(mean_squared_error(t, y_rs_pipe, squared=False)))
# print('rs_model', rs_model_pipe.best_score_)
# print('best params', rs_model_pipe.best_params_)
```

```
Accuracy score for regression:
Accuracy score on train 0.7063737587831955
RMSE score on train 1390799.5210448876
```

Mean Val Score: -0.22, Mean Train Score: 0.72



Mean Val Loss: 2384685.94, Mean Train Loss: 1349090.93

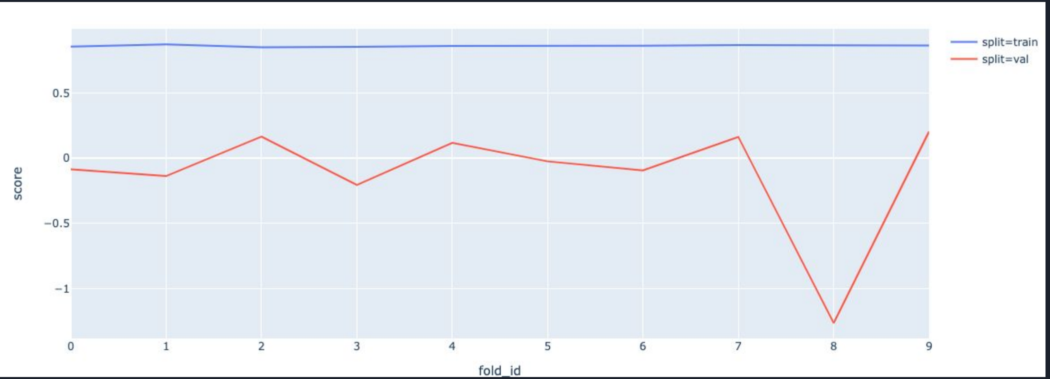


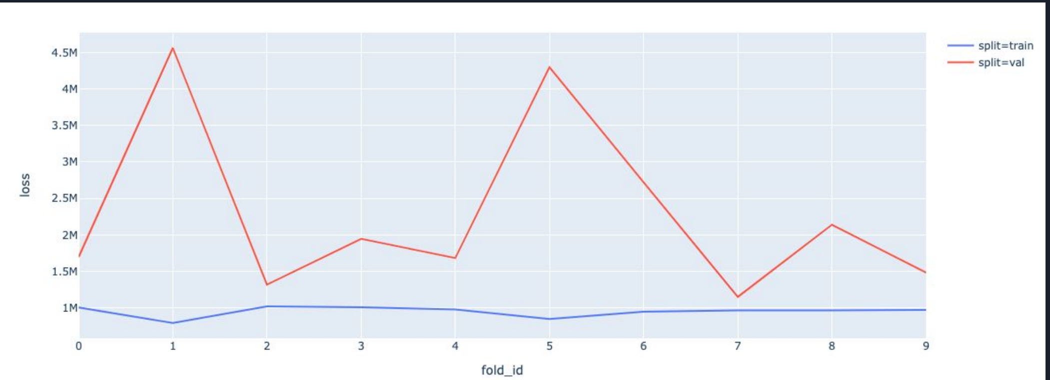
Random Forest

We used the basic model to get a feel for expected scores and loss values when using the Random Forest Regressor with default values using 10-fold cross validation

```
from sklearn.ensemble import RandomForestRegressor
forest_pipe  = pipeline.make_pipeline(ct, RandomForestRegressor(random_state=1)).fit(X, t)

val_score, val_loss, train_score, train_loss = get_cv_score_and_loss(X, t, forest_pipe, k=10, show_score_loss_graphs=True)
print(f'mean cv val score: {val_score:.2f}\nmean cv val loss {val_loss:.2f}')
print(f'mean cv train score: {train_score:.2f}\nmean cv train loss {train_loss:.2f}')
```

Random Forest - CV graphs






```
mean cv val score: -0.12
mean cv val loss 2300308.81
mean cv train score: 0.86
mean cv train loss 951519.74
```

Random Forest - default train

The prediction results when fitting and predicting with the default model

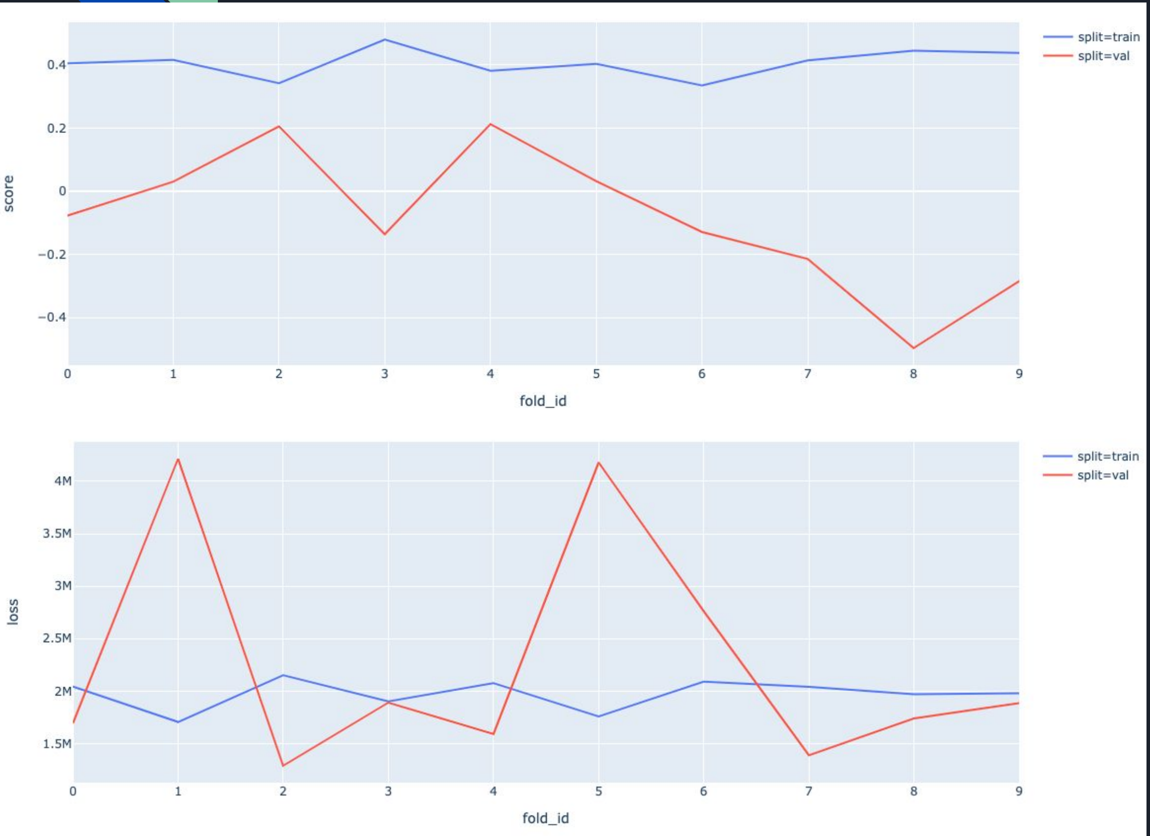
```
Accuracy score for regression:
Accuracy score on train 0.8513051936335301
RMSE score on train 989726.0460099587
```

Random Forest - Hyperparameter selection

Choosing best hyper parameters using GridSearchCV

```
[136] param_dist_forest = {
                    'max_depth': [5, 6, 7, 8, 9, 10, 15, 20, None],
                    'min_samples_leaf': range(1, 7, 1),
                    'min_samples_split': [2],
                    'n_estimators': range(1, 20, 1)
                 }

      forest_model = GridSearchCV(RandomForestRegressor(criterion='squared_error', random_state = 1), param_dist_forest, cv = 10).fit(X_encoded, t)
      print('Accuracy score for regression:')
      print('forest_model', forest_model.best_score_)
      print('best params', forest_model.best_params_)

      Accuracy score for regression:
      forest_model -0.0073600157435049106
      best params {'max_depth': 7, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_estimators': 6}
```

Random Forest - Hyperparameter selection CV




```
mean cv val score: -0.09
mean cv val loss 2263454.16
mean cv train score: 0.41
mean cv train loss 1971631.13
```

Random Forest - Hyperparameter selection Results

Chosen parameters:

max_depth=7
min_samples_leaf=5
min_samples_split=2
n_estimators=6

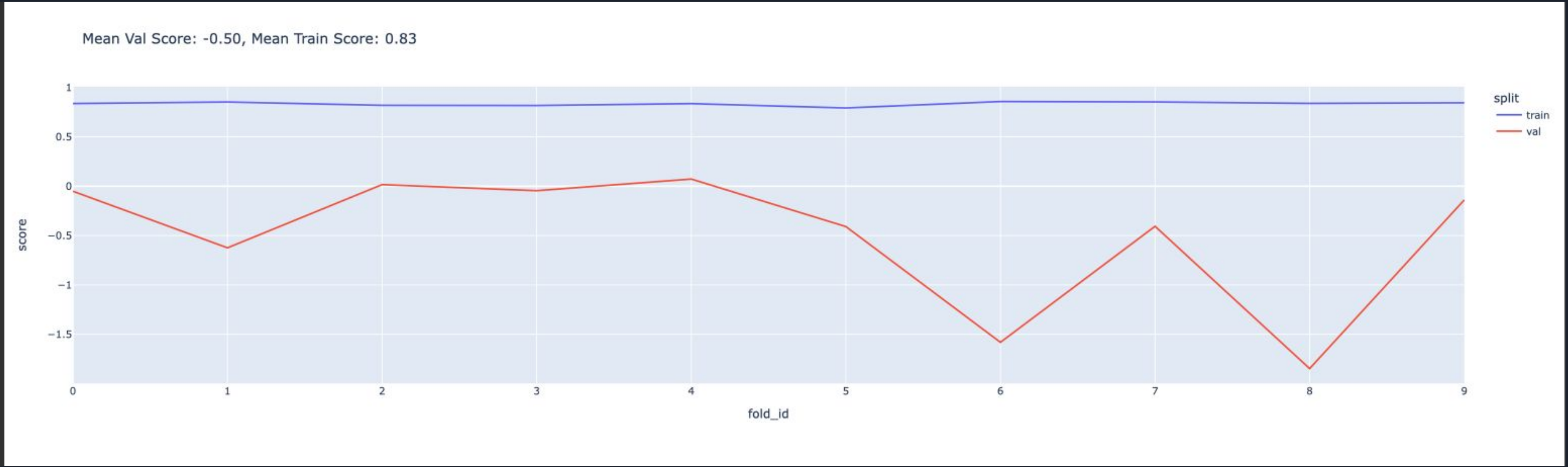
```
Accuracy score for regression:
Accuracy score on train 0.4505148801827643
RMSE score on train 1902589.3775937336
```

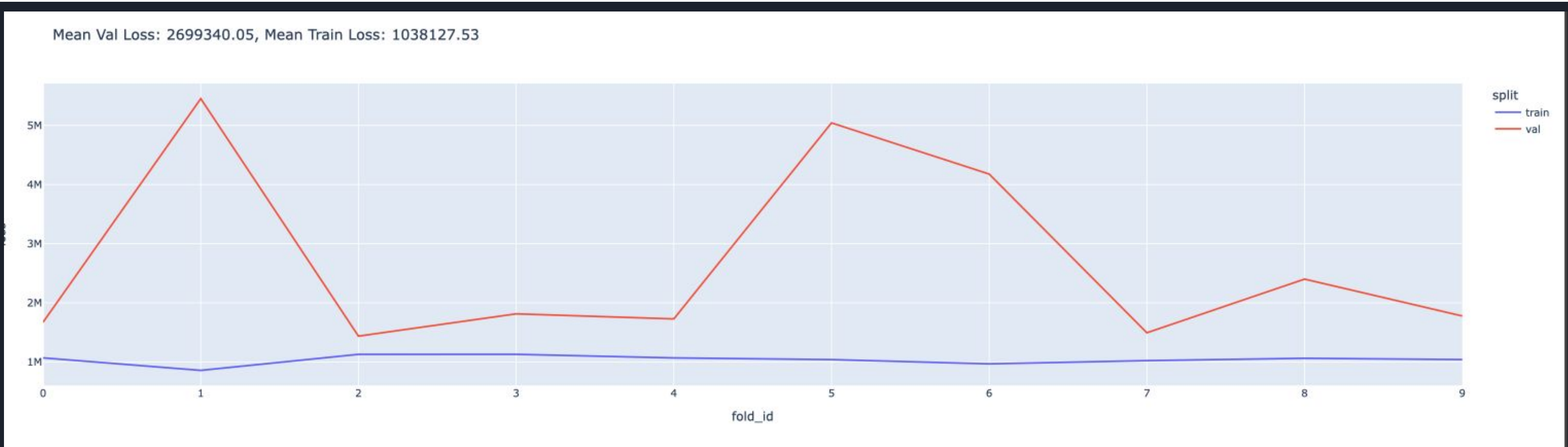
AdaBoost

An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

AdaBoost_model = pipeline.make_pipeline(ct, AdaBoostRegressor(DecisionTreeRegressor(max_depth=4), n_estimators=300, random_state=1, loss='square'))

val_score, val_loss, train_score, train_loss = get_cv_score_and_loss(X, t, AdaBoost_model, k=10, show_score_loss_graphs=True)
print(f'mean cv val score: {val_score:.2f}\nmean cv val loss {val_loss:.2f}')
print(f'mean cv val score: {train_score:.2f}\nmean cv val loss {train_loss:.2f}')
```

Mean cv val score: -0.50
Mean cv val loss: 2699340.05
Mean cv train score: 0.83
Mean cv val loss: 1038127.53


Mean Val Score: -0.50, Mean Train Score: 0.83
score
fold_id
split
train
val


Mean Val Loss: 2699340.05, Mean Train Loss: 1038127.53
fold_id
split
train
val

AdaBoost Train Result

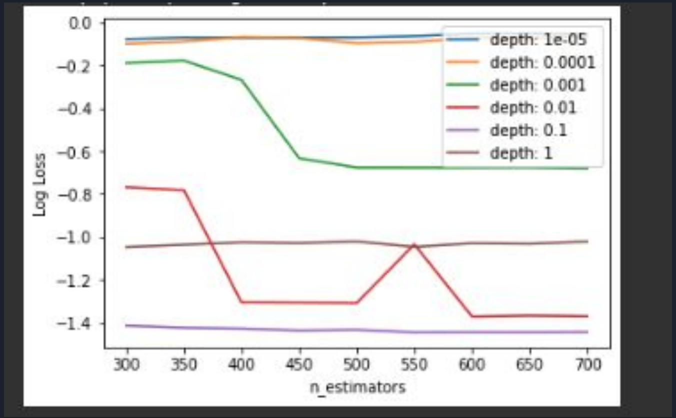
```
print('Accuracy score on train', AdaBoost_reg_original.score(X, t))
print('RMSE score on train',(mean_squared_error(t, y_train, squared=False)))
```

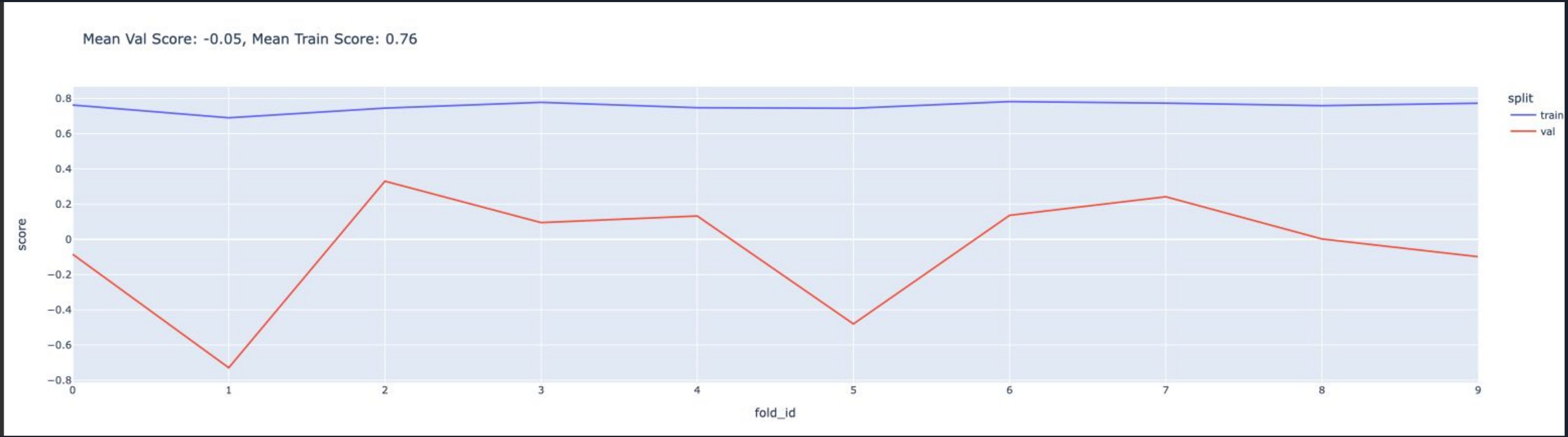
```
Accuracy score on train 0.8605435280405829
RMSE score on train 958487.4628856435
```

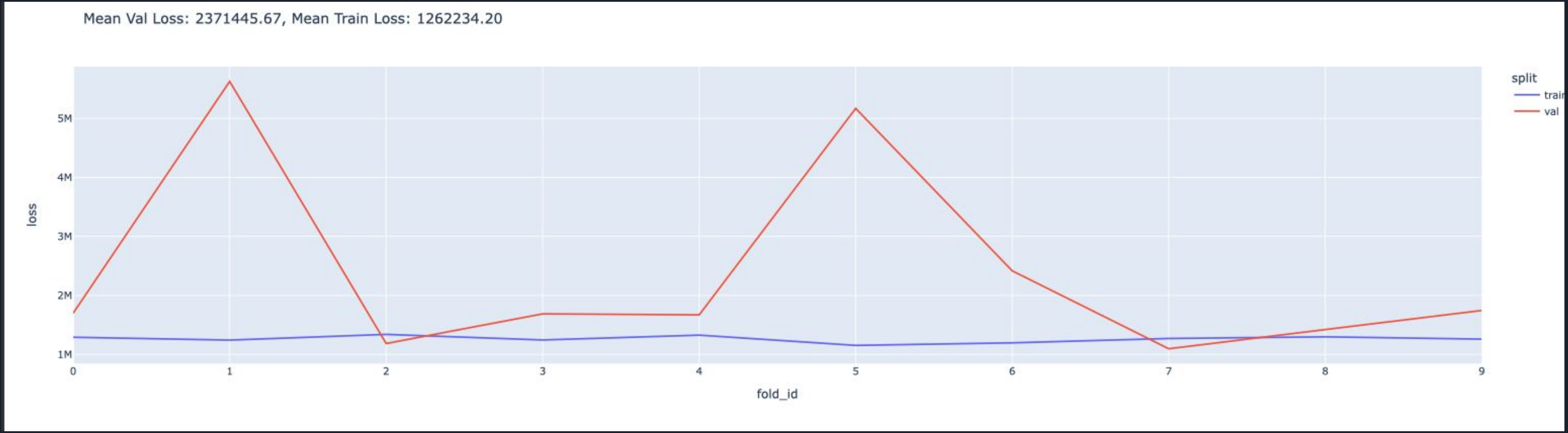
AdaBoost Hyperparameters

```
param_dist = {'n_estimators': [300,350,400,450,500,550,600, 650, 700],
              'learning_rate': [0.00001, 0.0001, 0.001 ,0.01,0.1, 1]
             }
ab_model = GridSearchCV(AdaBoostRegressor(DecisionTreeRegressor(max_depth=4), random_state=1, loss='square'), param_dist, cv=10).fit(X_encoded, t)
print('Accuracy score for regression:')
print('ab_model', ab_model.best_score_)
print('best params', ab_model.best_params_)
```

```
Accuracy score for regression:
ab_model -0.05400895673661517
best params {'learning_rate': 1e-05, 'n_estimators': 600}
```




Mean Val Score: -0.05, Mean Train Score: 0.76
score
fold_id
split
train
val


Mean Val Loss: 2371445.67, Mean Train Loss: 1262234.20
loss
fold_id
split
val

AdaBoost Train Hyperparameters Result

```
rs_model_pipe_Ada  = pipeline.make_pipeline(ct, AdaBoostRegressor(DecisionTreeRegressor(max_depth=4), random_state=1, loss='square',learning_rate= 1e-05, n_estimators=600)).fit(X, t)
print('Accuracy score for regression:')
y_rs_pipe = rs_model_pipe_Ada.predict(X)
print('Accuracy score on train', rs_model_pipe_Ada.score(X, t))
print('RMSE score on train',(mean_squared_error(t, y_rs_pipe, squared=False)))
```

```
Accuracy score for regression:
Accuracy score on train 0.7074899962165696
RMSE score on train 1388153.4006967554
```

Results Comparison

	SGDRegressor	SGDRegressor tuned	XGBoost	XGBoost tuned	AdaBoost	AdaBoost tuned	Random Forest	Random Forest tuned
R2	0.33	0.18	0.90	0.70	0.80	0.70	0.85	0.45
RMSE	2129869.67	2317663.68	785885.47	1390799.52	1130760.75	1388153.40	989726.04	1902589.37

Test

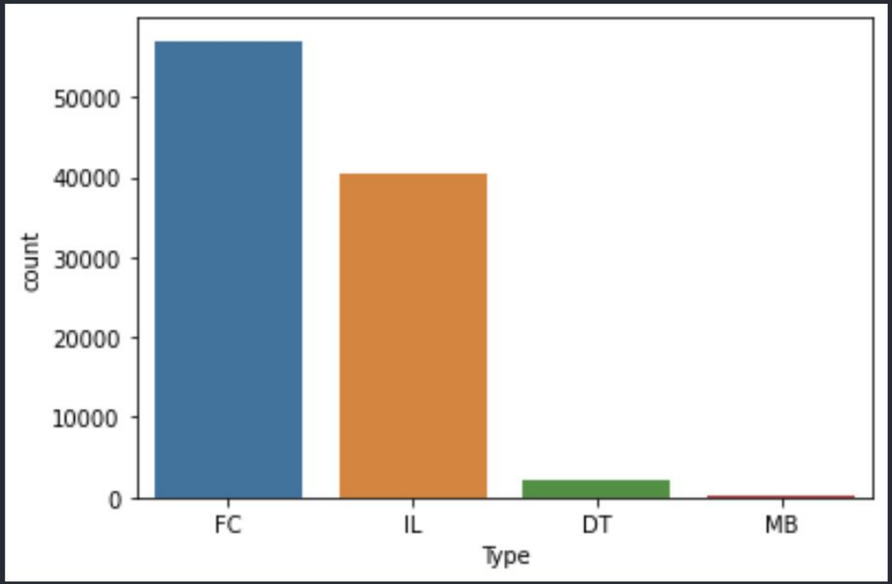
100000 samples in the test > 137 samples in the train

	Id	Open Date	City	City Group	Type	P1	P2	P3	P4	P5	...	P28	P29	P30	P31	P32	P33	P34	P35	P36	P37
0	0	01/22/2011	Niğde	Other	FC	1	4.0	4.0	4.0	1	...	2.0	3.0	0	0	0	0	0	0	0	0
1	1	03/18/2011	Konya	Other	IL	3	4.0	4.0	4.0	2	...	1.0	3.0	0	0	0	0	0	0	0	0
2	2	10/30/2013	Ankara	Big Cities	FC	3	4.0	4.0	4.0	2	...	2.0	3.0	0	0	0	0	0	0	0	0
3	3	05/06/2013	Kocaeli	Other	IL	2	4.0	4.0	4.0	2	...	2.0	3.0	0	4	0	0	0	0	0	0
4	4	07/31/2013	Afyonkarahisar	Other	FC	2	4.0	4.0	4.0	1	...	5.0	3.0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99995	99995	01/05/2000	Antalya	Other	FC	5	5.0	4.0	4.0	2	...	2.0	3.0	0	0	0	0	0	0	0	0
99996	99996	07/18/2011	Niğde	Other	IL	1	2.0	4.0	3.0	1	...	1.0	3.0	5	0	0	0	4	0	0	0
99997	99997	12/29/2012	İstanbul	Big Cities	IL	4	5.0	4.0	4.0	1	...	4.0	1.0	5	5	3	2	4	4	4	2
99998	99998	10/12/2013	İstanbul	Big Cities	FC	12	7.5	6.0	6.0	4	...	7.5	5.0	0	0	0	4	0	0	0	0
99999	99999	10/05/2010	İstanbul	Big Cities	IL	2	5.0	4.0	4.0	2	...	5.0	3.0	0	2	0	2	2	4	2	0

100000 rows × 42 columns

Test Data

Test set don’t have empty values, how ever there is 1 for type of restaurant.




```
[ ] test_df_cp.replace('MB', 'DT', inplace=True)
```

Test Results Compreation

	SGDRegressor	**SGDRegressor tuned**	XGBoost	XGBoost tuned	AdaBoost	AdaBoost tuned	Random Forest	**Random Forest tuned**
RMSE-Public	2117966.07873	**1826302.00535**	2504617.03774	2265356.53008	2038560.943049	1911685.27160	2081712.88515	**1768327.26584**
RMSE-Private	1928312.03781	**1745662.51973**	2340390.44798	2294269.34289	2060857.15330	1918850.87706	1955265.32349	**1870204.72282**