CSE204: Project

Accurately forecast demand for grocery retailers, to reduce food waste and increase availability.

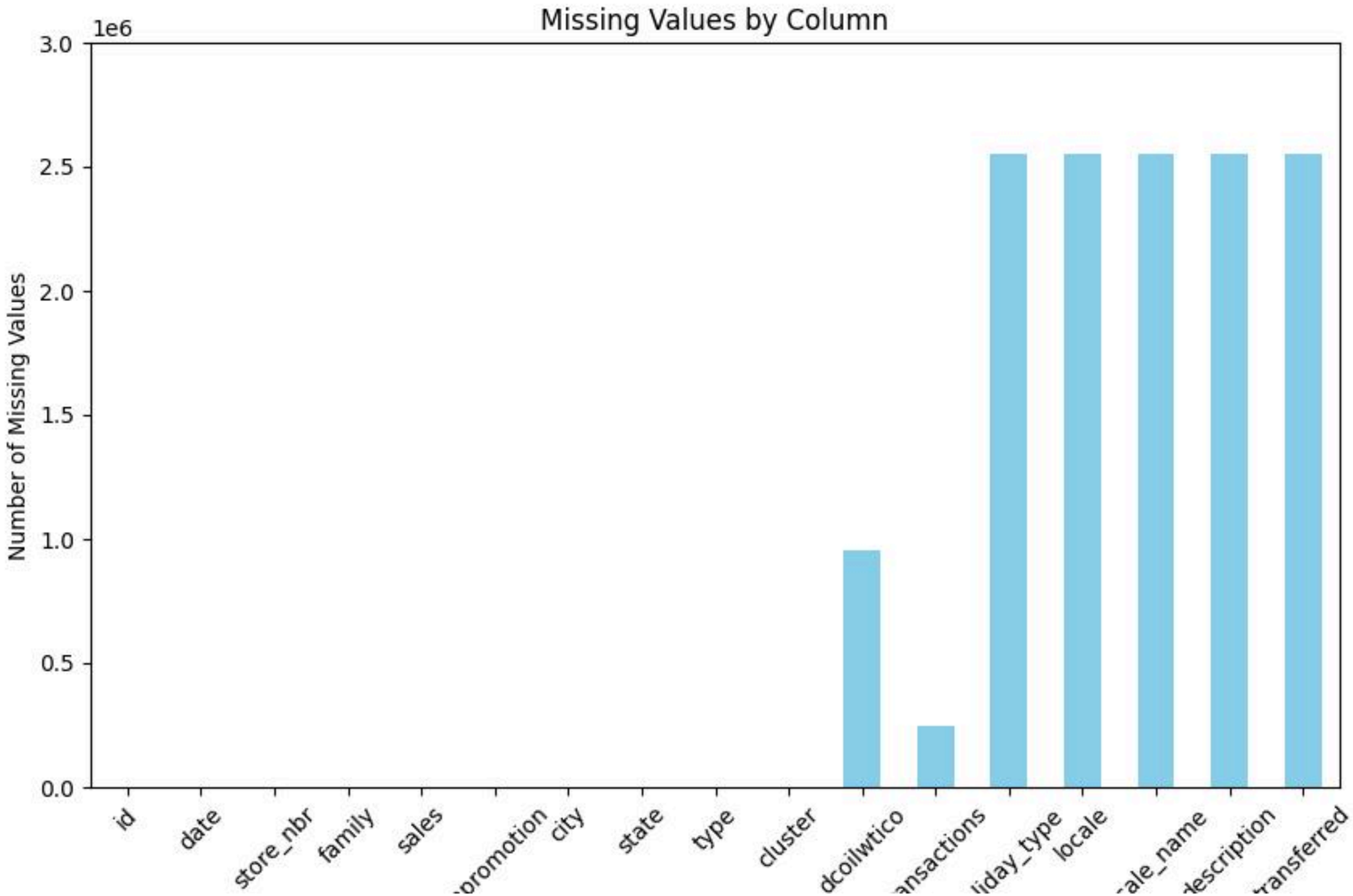
By Dan Suissa, Adan Fhima, Maximilien Bruck

Exploratory Data Analysis

The Datasets

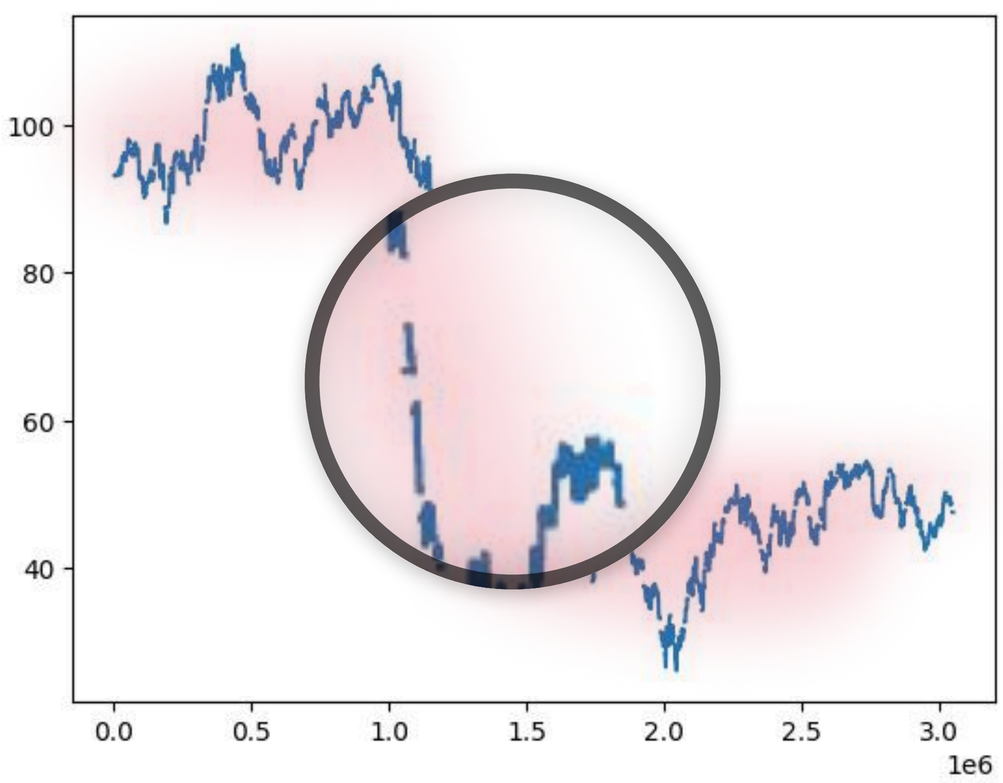
Sales.csv: *data with time series and sales*	date	store_nbr	sales	family	onpromotion
Transaction.csv: *records of transactions made on specific dates*	date	store_nbr	transactions		
Stores.csv: *metadata about stores*	store_nbr	city	state	type	cluster
Holidays.csv: *information about holidays and events*	date	type	locale	description	
Oil.csv: *daily oil prices*	date	oil_price			

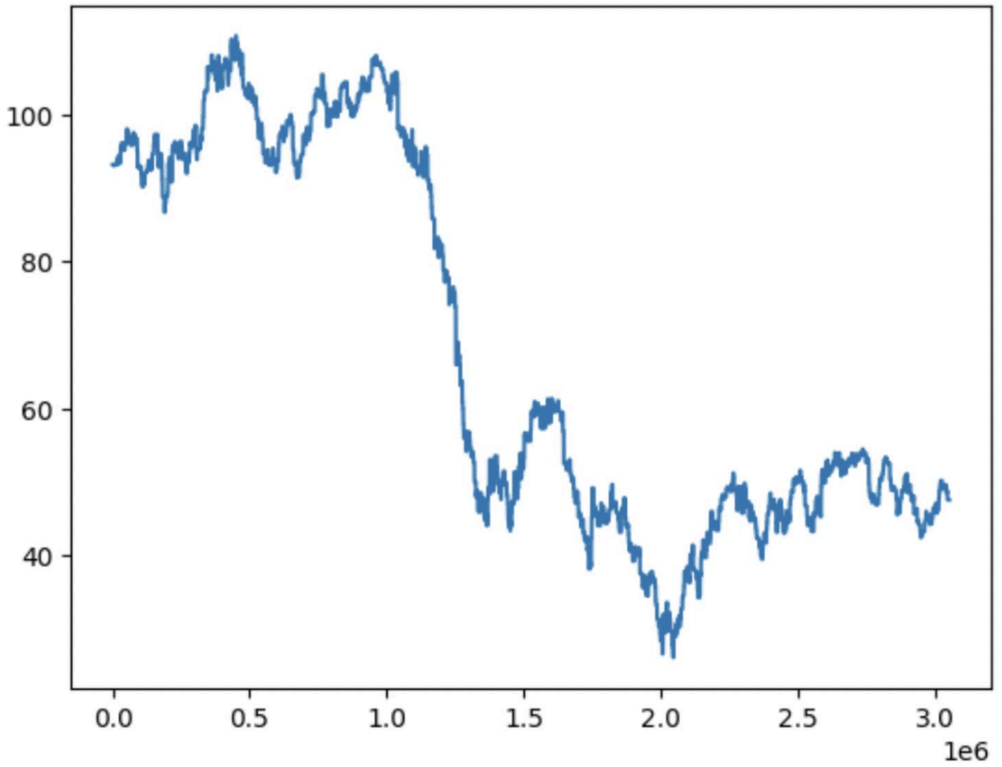
Missing Values



Missing values: Oil Price

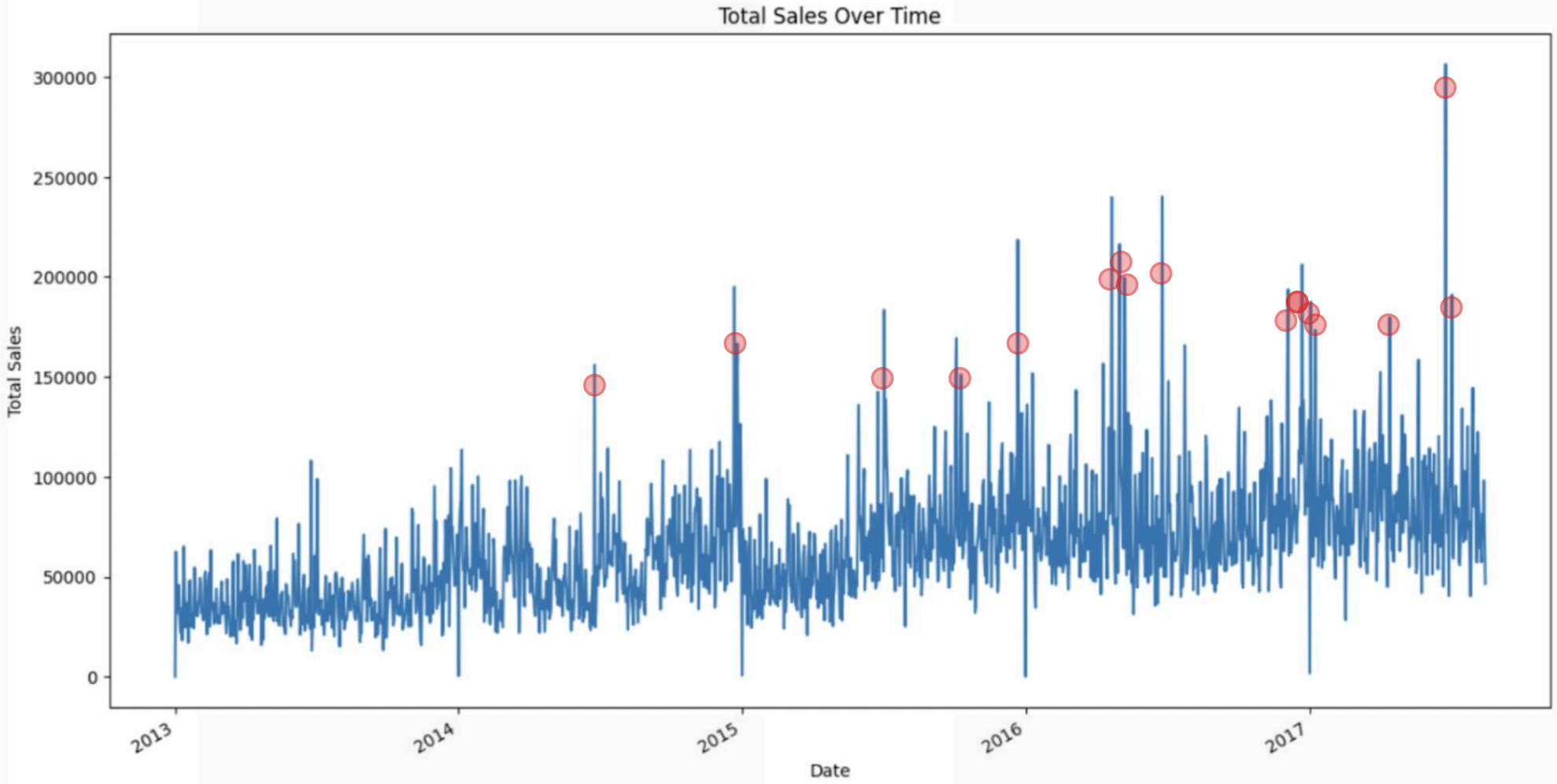
Before — .bfill() — After



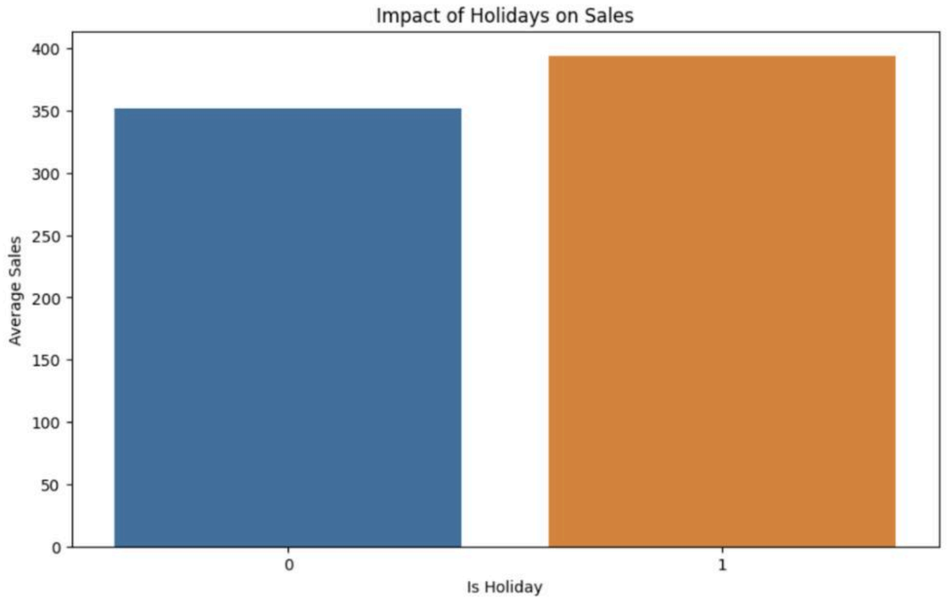


Exploratory Data Analysis

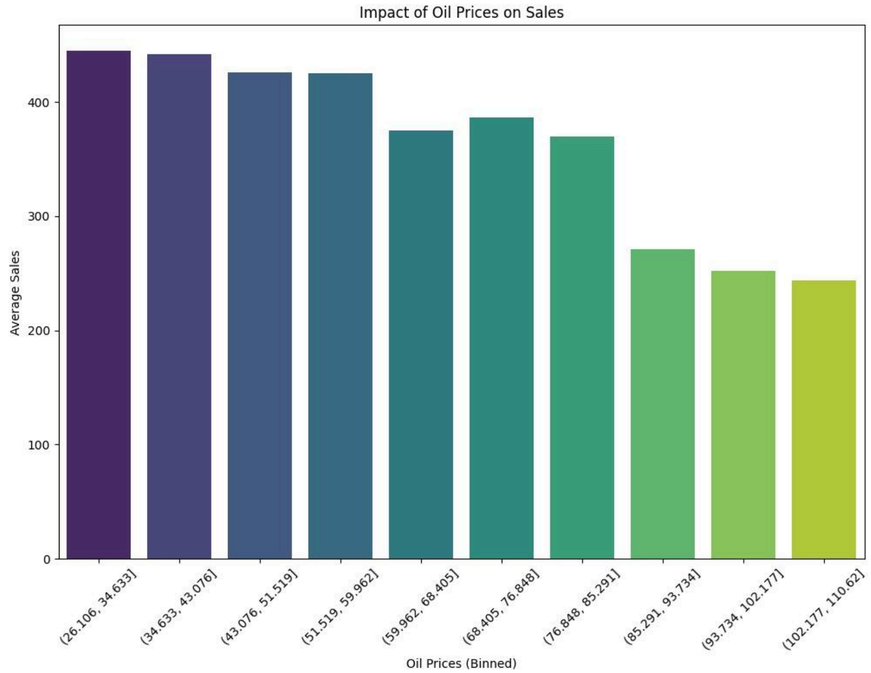
Sales over time



Correlation Matrix



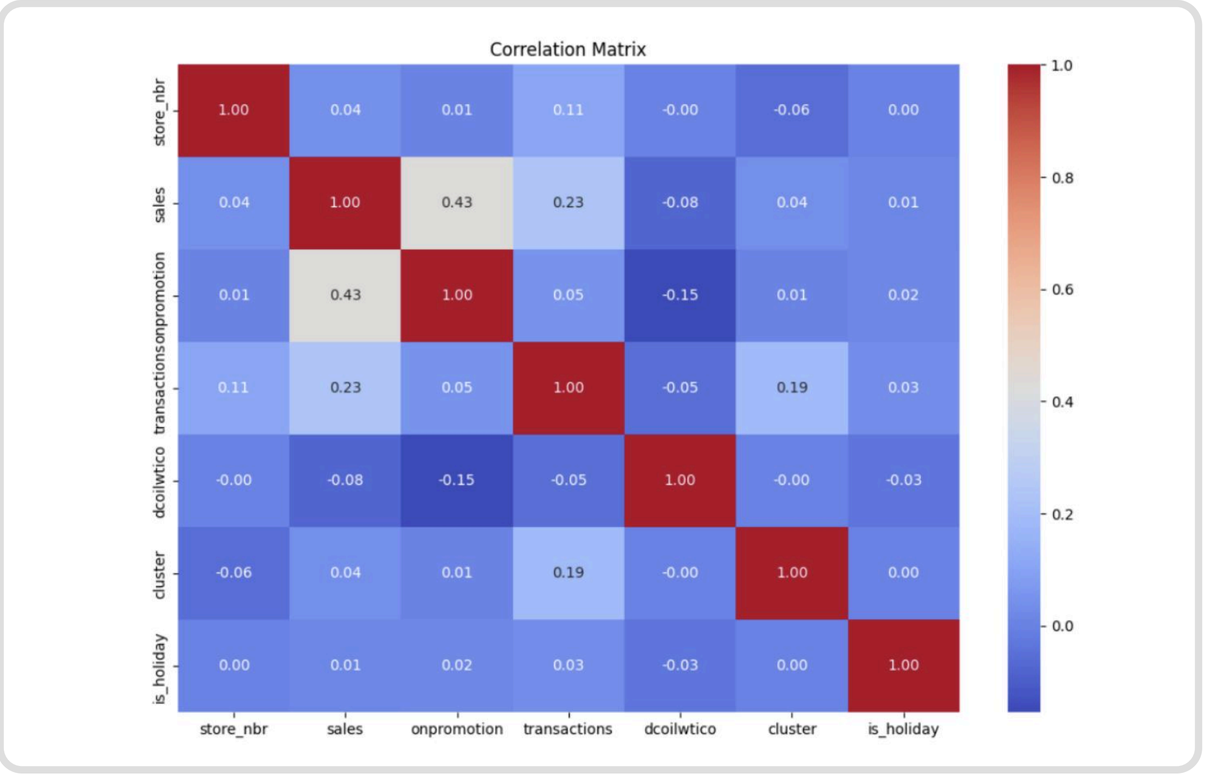
Holidays vs Sales

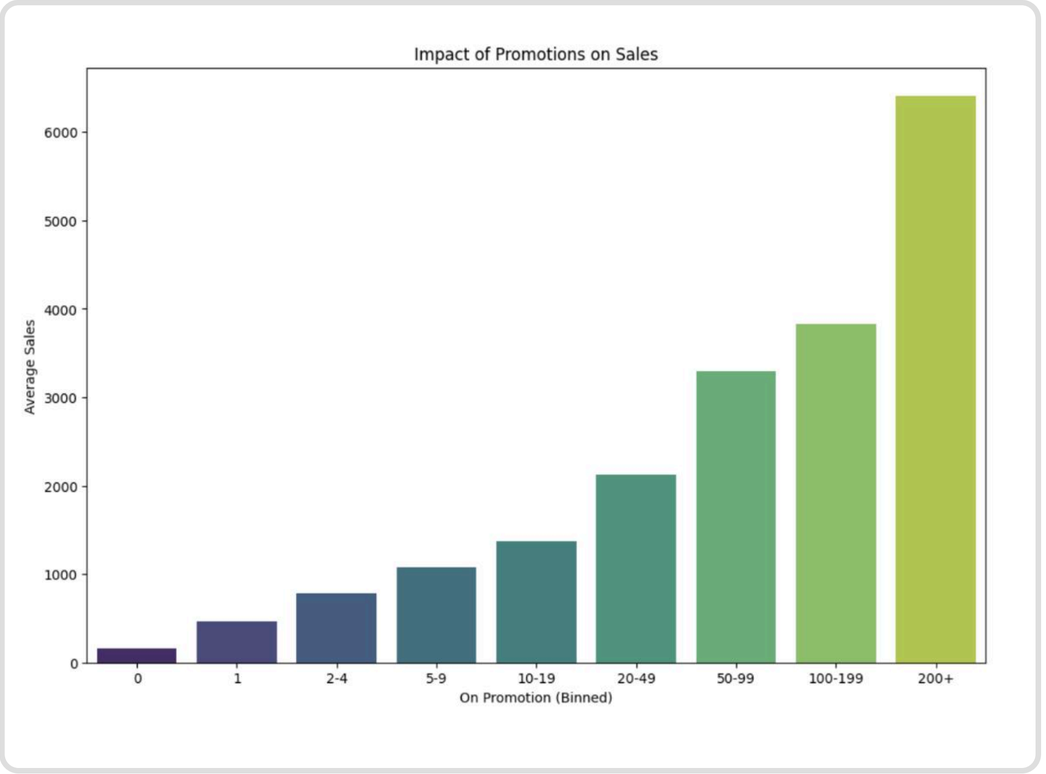


Oil Prices vs Sales

Exploratory Data Analysis

Correlation Matrix





Preprocessing data

Transferred holidays


```python
# Defining the function to adjust transferred holidays
def adjust_and_remove_transferred_holidays(holidays_df):
    for idx, row in holidays_df[holidays_df['holiday_type'] == 'Transfer'].iterrows():
        corresponding_row = holidays_df[(holidays_df['description'] == row['description']) &
                                        (holidays_df['holiday_type'] != 'Transfer')]
        if not corresponding_row.empty:
            holidays_df.at[idx, 'date'] = corresponding_row.iloc[0]['date']

    holidays_df.drop_duplicates(inplace=True)

    # Removing all rows where 'holiday_type' is 'Transfer'
    holidays_df = holidays_df[holidays_df['holiday_type'] != 'Transfer']

    return holidays_df

# Applying the function to adjust and remove transferred holidays
```

Date Features

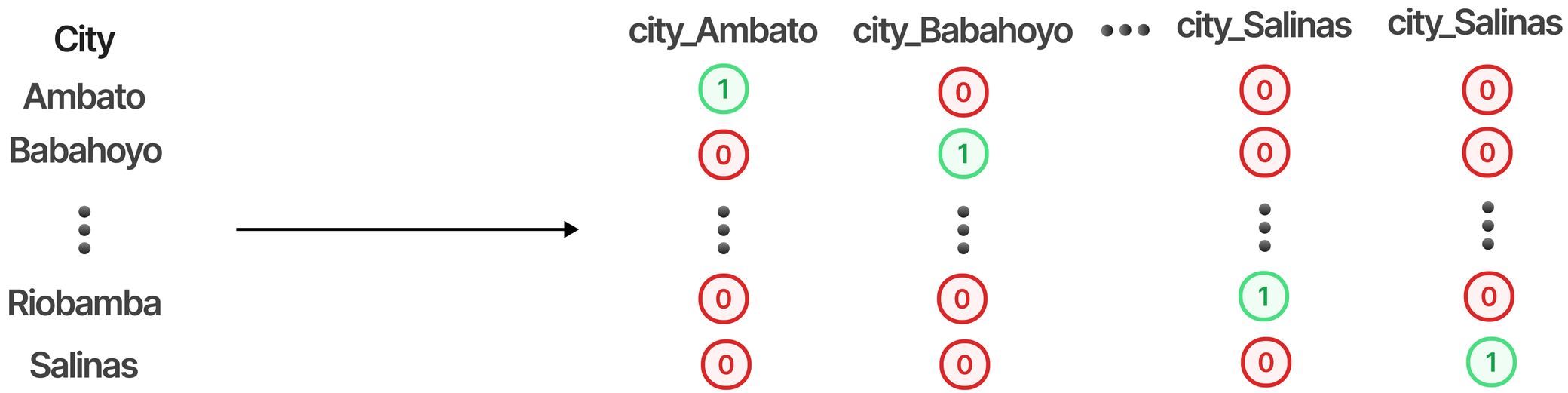
store_nbr	sales	•••	date	is_weekend	month	day_of_week
12	34		23/11/2017	1	11	6
7	22		11/09/2018	0	6	5
6	19		12/08/2018	0	6	5

```
df['date'] = pd.to_datetime(df['date'])
df['day_of_week'] = df['date'].dt.dayofweek
df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)
df['month'] = df['date'].dt.month

df = df.set_index('date')
df.head()

#Let's get a preview for our dataset from now
```

Encoding




```
cat_encoder = OneHotEncoder(sparse_output=False)
data_cat_1hot = cat_encoder.fit_transform(data_cat_df)

data_cat_df = pd.DataFrame(data_cat_1hot, columns=cat_encoder.get_feature_names_out())
```

Setting error metrics

Mean Absolute Error

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```python
from sklearn.metrics import mean_squared_error,mean_absolute_error

def checkModelPerformance(model):
    model.fit(x_train.values, y_train.values)

    pred = model.predict(x_test.values)

    print("mean_squared_error: ",np.sqrt(mean_squared_error(y_test.values, pred)))
    print("mean_absolute_error: ", np.sqrt(mean_absolute_error(y_test.values, pred)))
```

Models study

Comparing regression models

```python
from sklearn.linear_model import MultiTaskLassoCV

lasso_model_cv = MultiTaskLassoCV(alphas=[0.01, 0.1, 1])
lasso_model_cv.fit(x_train.values, y_train.values)
y_pred_lasso = lasso_model_cv.predict(x_test.values)
mse_lasso = mean_squared_error(y_test.values, y_pred_lasso)
mae_lasso = mean_absolute_error(y_test.values, y_pred_lasso)
```

Mean squared error	0.8593
Mean absolute error	0.6713

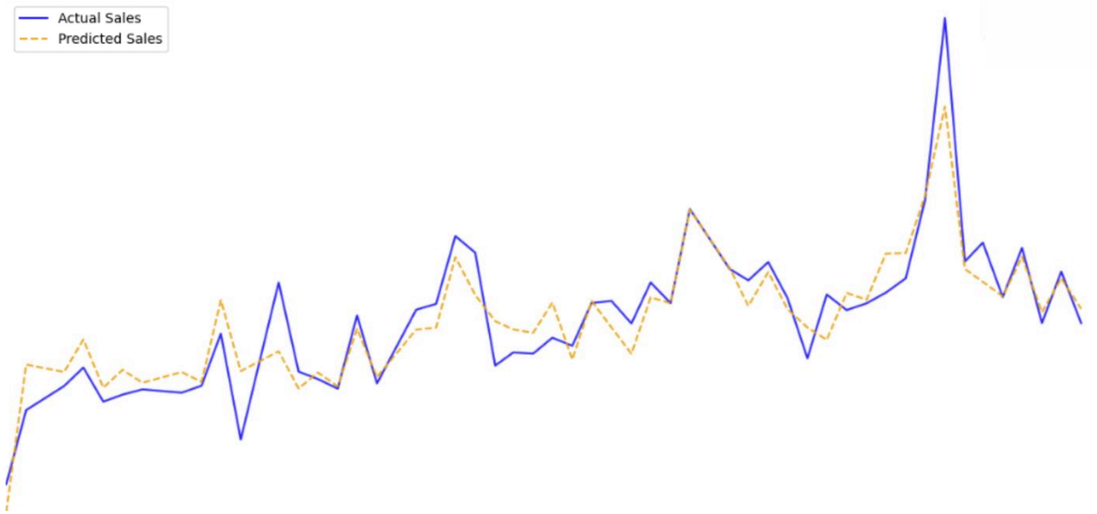
```python
from sklearn.linear_model import LinearRegression
checkModelPerformance(LinearRegression())
```

Mean squared error	0.6177
Mean absolute error	0.5335

```python
from sklearn.linear_model import RidgeCV


ridge_model_cv = RidgeCV(alphas=[0.01, 0.1, 1])
ridge_model_cv.fit(x_train.values, y_train.values)
y_pred_ridge = ridge_model_cv.predict(x_test.values)
mse_ridge = mean_squared_error(y_test.values, y_pred_ridge)
mae_ridge = mean_absolute_error(y_test.values, y_pred_ridge)
```

Mean squared error	0.6177
Mean absolute error	0.5334



Models study

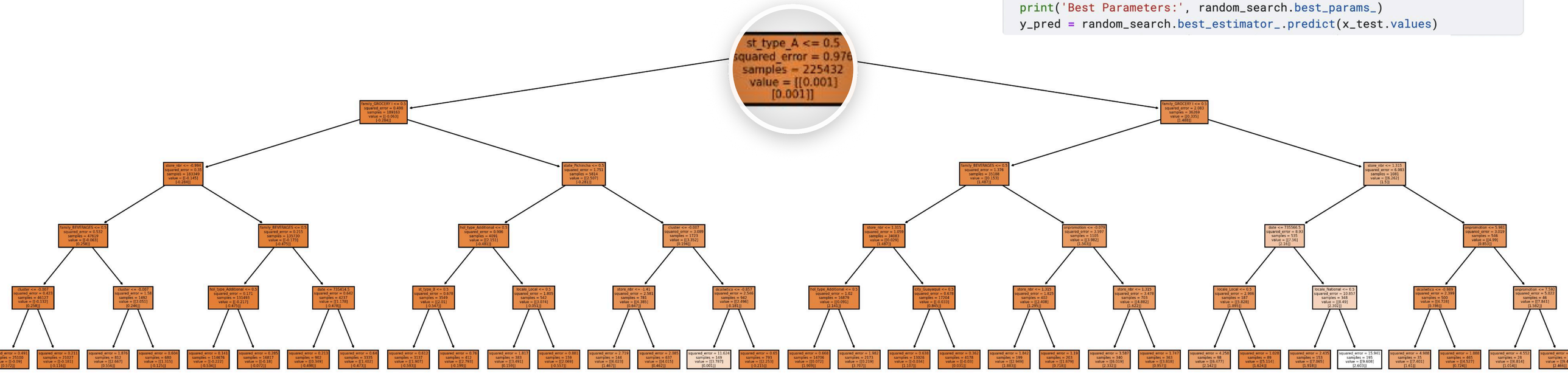
Decision Tree

Mean squared error	0.4458
Mean absolute error	0.3352

```python
from scipy.stats import randint
from sklearn.model_selection import RandomizedSearchCV


hyperparameters = {
    'max_depth': randint(1, 6),
    'min_samples_leaf': randint(5, 20)
}

decision_tree = DecisionTreeRegressor(random_state=42)
random_search = RandomizedSearchCV(
    decision_tree,
    param_distributions=hyperparameters,
    n_iter=10,
    cv=3,
    scoring='neg_mean_squared_error',
    random_state=42,
    n_jobs=-1
)
random_search.fit(x_train.values, y_train.values)
print('Best Parameters:', random_search.best_params_)
y_pred = random_search.best_estimator_.predict(x_test.values)
```



Models study

Random Forest

```
param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},]

forest_reg = RandomForestRegressor(random_state=42)

grid_search = GridSearchCV(forest_reg, param_grid, cv=5, scoring='neg_mean_squared_error', return_train_score=True)
grid_search.fit(x_train.values, y_train.values)
print('Best Parameters:', grid_search.best_params_)
y_pred = grid_search.predict(x_test.values)
mse = mean_squared_error(y_test.values, y_pred)
mae = mean_absolute_error(y_test.values, y_pred)
print("Mean Squared Error (MSE) For Random Forest:", mse_rf)
print("Mean Absolute Error (MAE):", mae_rf)
```