DSC 630 - Predictive Analytics

Term Project - Summer 2024

David Berberena | Brian Mann

In [1]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, StackingRegressor
from sklearn.base import BaseEstimator, RegressorMixin
import statsmodels.api as sm
```

In [2]:
```
# hide warnings
import warnings
warnings.filterwarnings('ignore')
from sklearn.exceptions import ConvergenceWarning
```

Milestone 3 - Preliminary Analysis

In [3]:
```
# dataset
url = 'https://www.kaggle.com/datasets/mrsimple07/restaurants-revenue-prediction'
```

Step 1: Reading and Describing the Data

In [4]:
```
# reazd in the data and show the first few rows
df = pd.read_csv('restaurant_revenue.csv')
df.head()
```

Out[4]:

	Number_of_Customers	Menu_Price	Marketing_Spend	Cuisine_Type	Average_Customer_Spending	Promotions	Reviews	Monthly_R
0	61	43.117635	12.663793	Japanese	36.236133	0	45	350.
1	24	40.020077	4.577892	Italian	17.952562	0	36	221.
2	81	41.981485	4.652911	Japanese	22.600420	1	91	326.
3	70	43.005307	4.416053	Italian	18.984098	1	59	348.
4	30	17.456199	3.475052	Italian	12.766143	1	30	185.

In [5]:
```
# get the size and shape
df.shape
```

Out[5]: (1000, 8)

In [6]:
```
# check for missing values
df.isna().sum()
```

Out[6]:
```
Number_of_Customers         0
Menu_Price                  0
Marketing_Spend             0
Cuisine_Type                0
Average_Customer_Spending   0
Promotions                  0
Reviews                     0
Monthly_Revenue             0
dtype: int64
```

In [7]:
```
# assess common stats for numerical variables
df.describe()
```

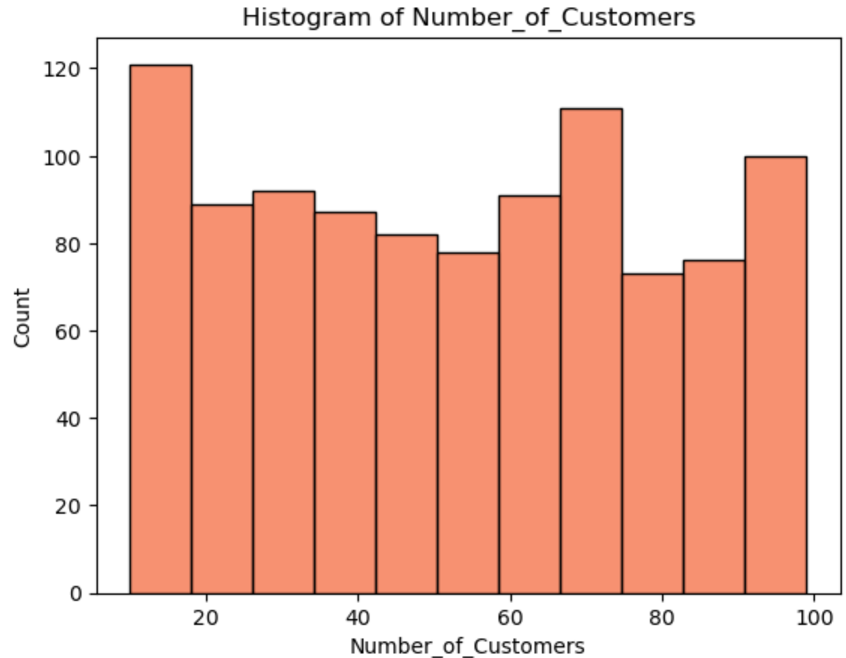
Out[7]:

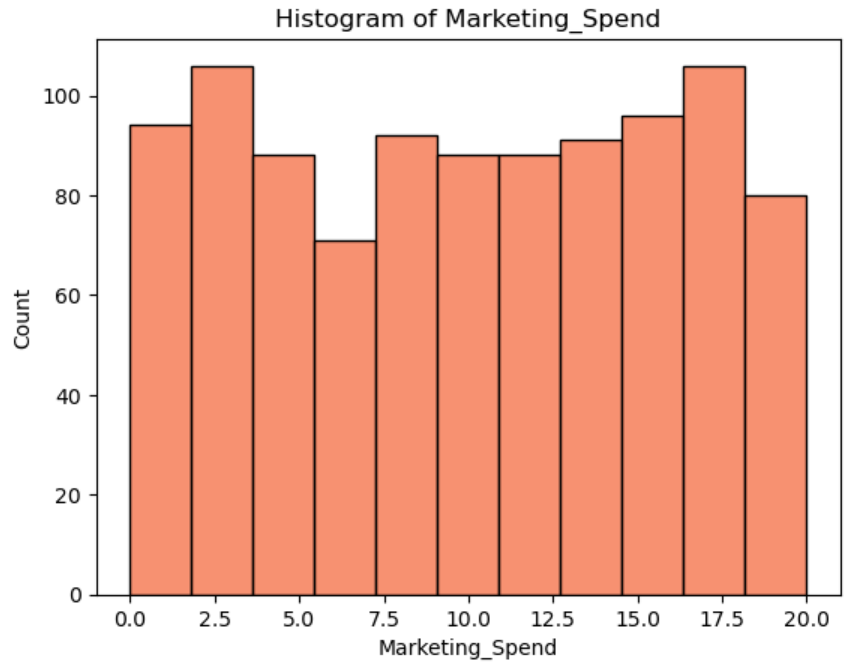
	Number_of_Customers	Menu_Price	Marketing_Spend	Average_Customer_Spending	Promotions	Reviews	Monthly_Revenu
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	53.271000	30.219120	9.958726	29.477085	0.497000	49.837000	268.72417
std	26.364914	11.278760	5.845586	11.471686	0.500241	29.226334	103.98295
min	10.000000	10.009501	0.003768	10.037177	0.000000	0.000000	-28.97780
25%	30.000000	20.396828	4.690724	19.603041	0.000000	24.000000	197.10364
50%	54.000000	30.860614	10.092047	29.251365	0.000000	50.000000	270.21396
75%	74.000000	39.843868	14.992436	39.553220	1.000000	76.000000	343.39579
max	99.000000	49.974140	19.994276	49.900725	1.000000	99.000000	563.38133

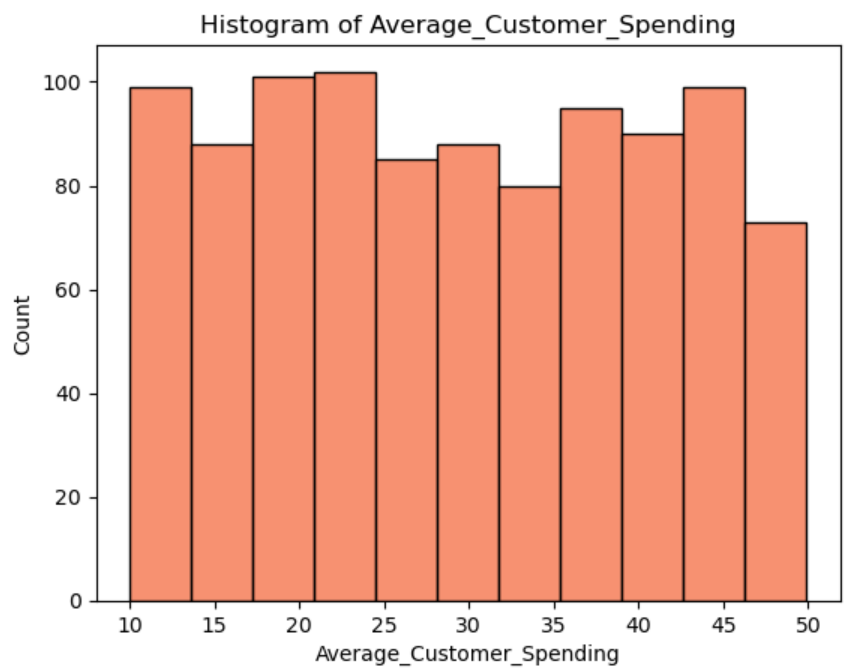
Step 2: Histograms and Bar Charts

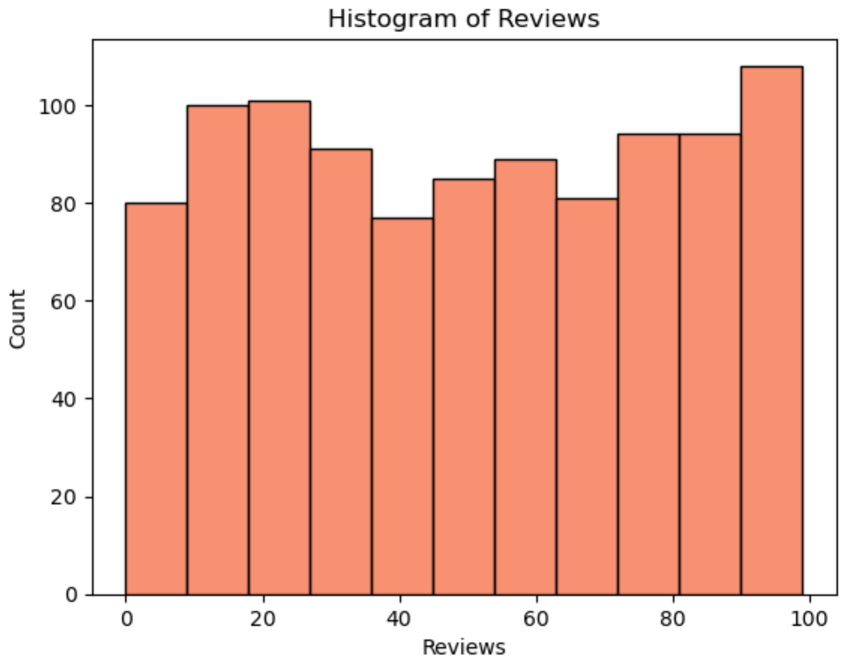
In [8]:
```
# this function takes in a df and a column and prints out a histogram
def make_histogram(data, col):
    sns.histplot(data[col], color='#f56c42')
    plt.title(f'Histogram of {col}')
    plt.show()
```

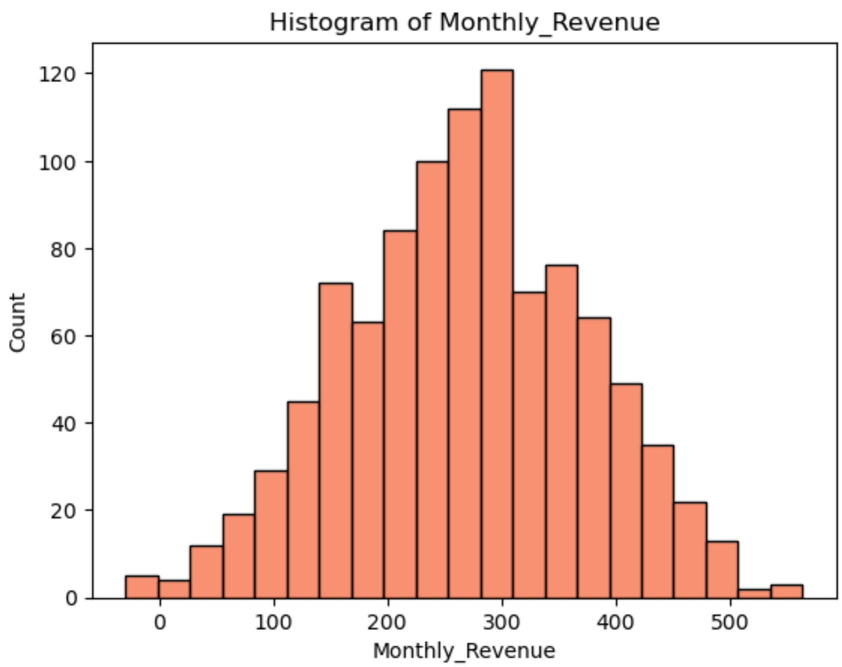
In [9]:
```python
# print out histograms for each of the primary numerical variables
for col in df.describe().drop(columns='Promotions').columns:
    make_histogram(df, col)
```




Histogram of Marketing_Spend
Count
0
20
40
60
80
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Marketing_Spend

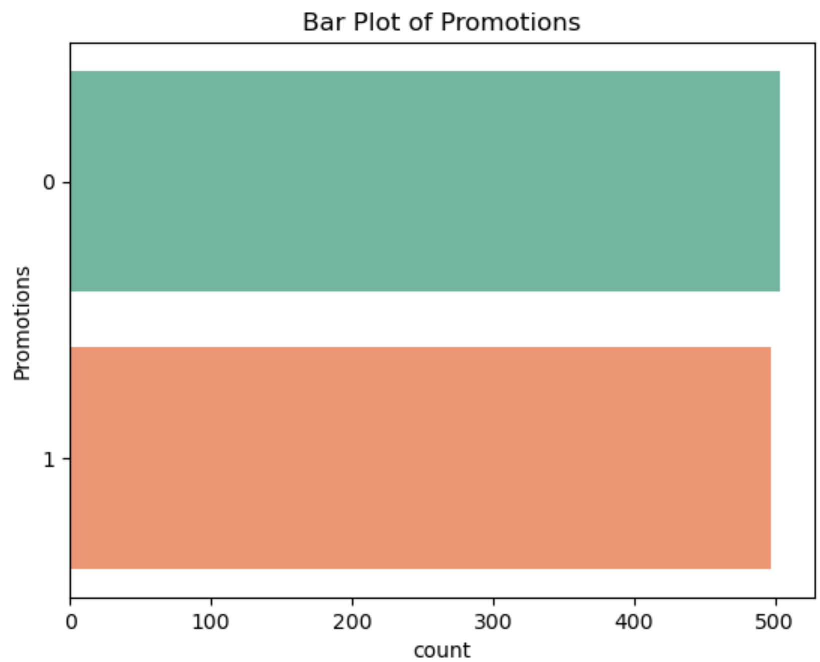

Histogram of Average_Customer_Spending
Count
0
20
40
60
80
100
10
15
20
25
30
35
40
45
50
Average_Customer_Spending

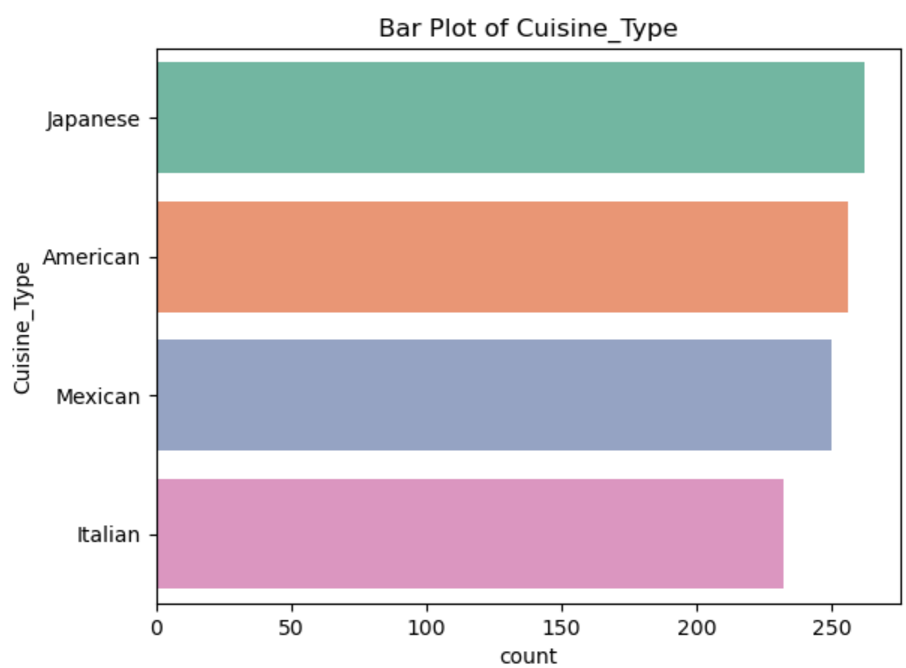




In [10]:
```
# this function takes in a df and a column and prints out a horizontal bar
# plot of the counts
def count_bar(data, col):
    counts = data[col].value_counts().reset_index()
    sns.barplot(data=counts, y=col, x='count', orient='h', palette='Set2')
    plt.title(f'Bar Plot of {col}')
    plt.show()
```

In [11]:
```
# print out bar plots for the counts of promotions and cusine type
for col in ['Promotions', 'Cuisine_Type']:
    count_bar(df, col)
```





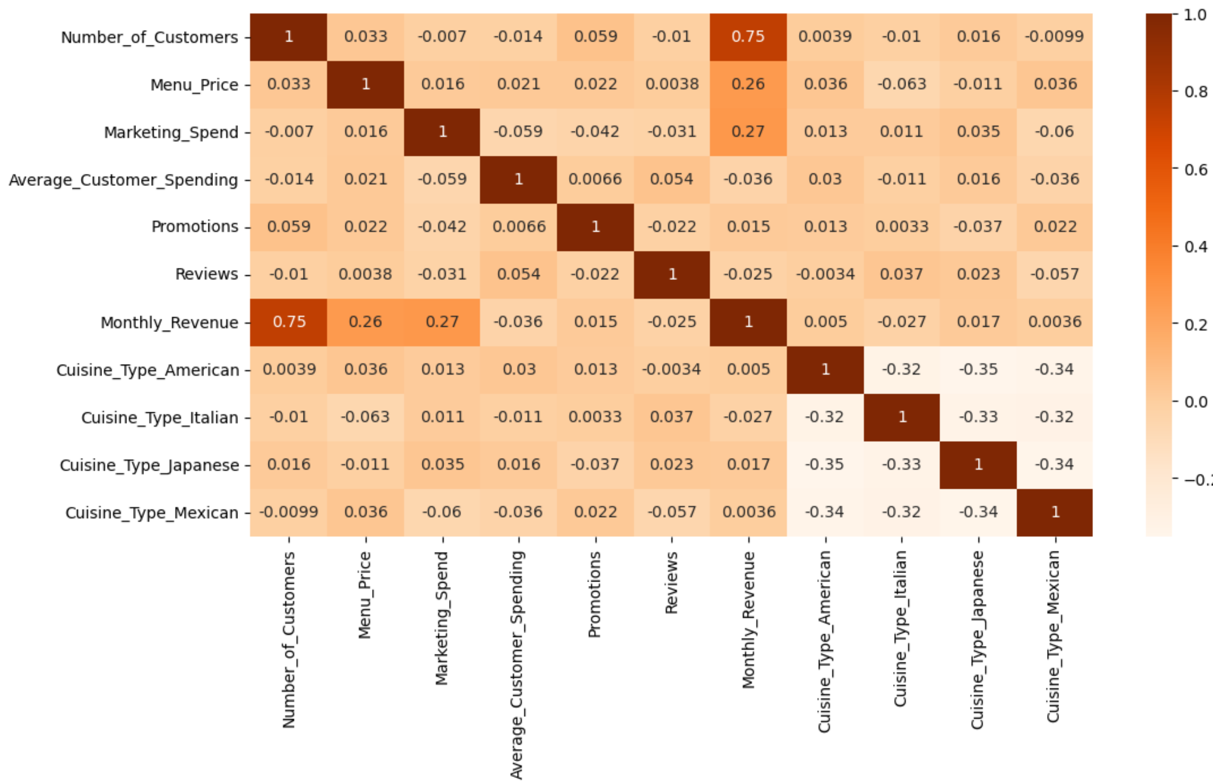
Step 3: Correlation and Scatterplots

In [12]:
```python
# create dummy variables for cuisine type and make it its own df
dum = pd.get_dummies(df, columns=['Cuisine_Type'])
dum.head()
```

Out[12]:

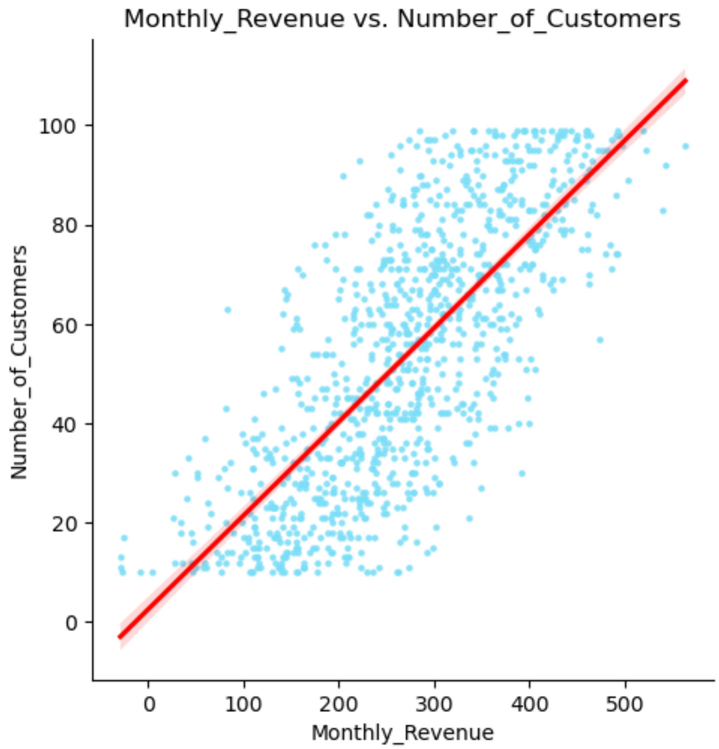
	Number_of_Customers	Menu_Price	Marketing_Spend	Average_Customer_Spending	Promotions	Reviews	Monthly_Revenue	Cuisin
0	61	43.117635	12.663793	36.236133	0	45	350.912040	
1	24	40.020077	4.577892	17.952562	0	36	221.319091	
2	81	41.981485	4.652911	22.600420	1	91	326.529763	
3	70	43.005307	4.416053	18.984098	1	59	348.190573	
4	30	17.456199	3.475052	12.766143	1	30	185.009121	

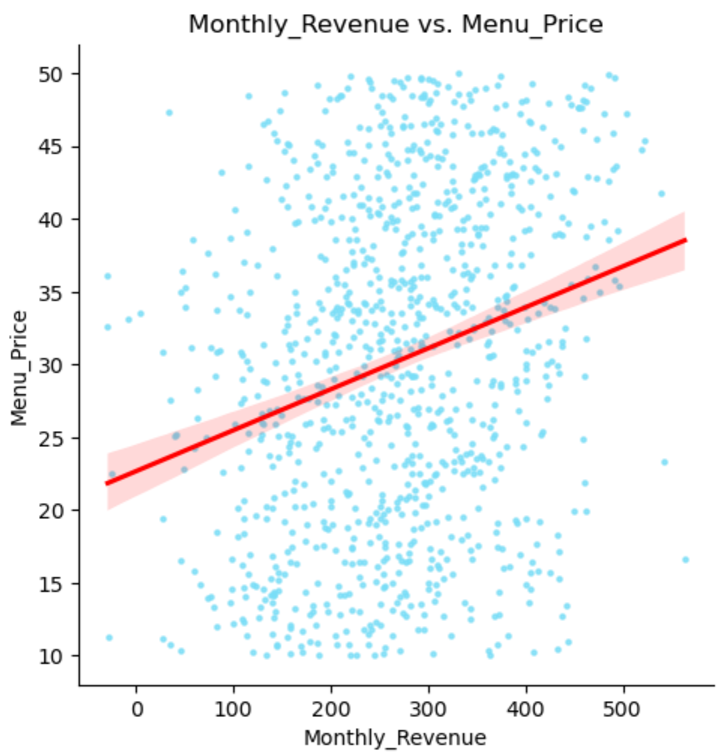
In [13]:
```python
# find the correlation among all of the variales in the df
corrs = dum.corr()
plt.figure(figsize=(12,6))
# plot a heatmap of the correlations
sns.heatmap(corrs, cmap="Oranges", annot=True)
plt.show()
```



In [14]:
```python
# this function takes in a df and two columns, then prints out a scatter plot
# with a line of best fit containing margins of error
def scatter_with_fit(data, col1, col2):
    sns.lmplot(data=data, x=col1, y=col2, line_kws={'color': 'red'},
               scatter_kws={'color': '#79def7', 's': 5})
    plt.title(f'{col1} vs. {col2}')
    plt.show()
```

In [15]:
```python
# print out scatterplots with best fine lines for the variables with
# high correlations with monthly revenue
for col in ['Number_of_Customers', 'Menu_Price', 'Marketing_Spend']:
    scatter_with_fit(df, 'Monthly_Revenue', col)
```

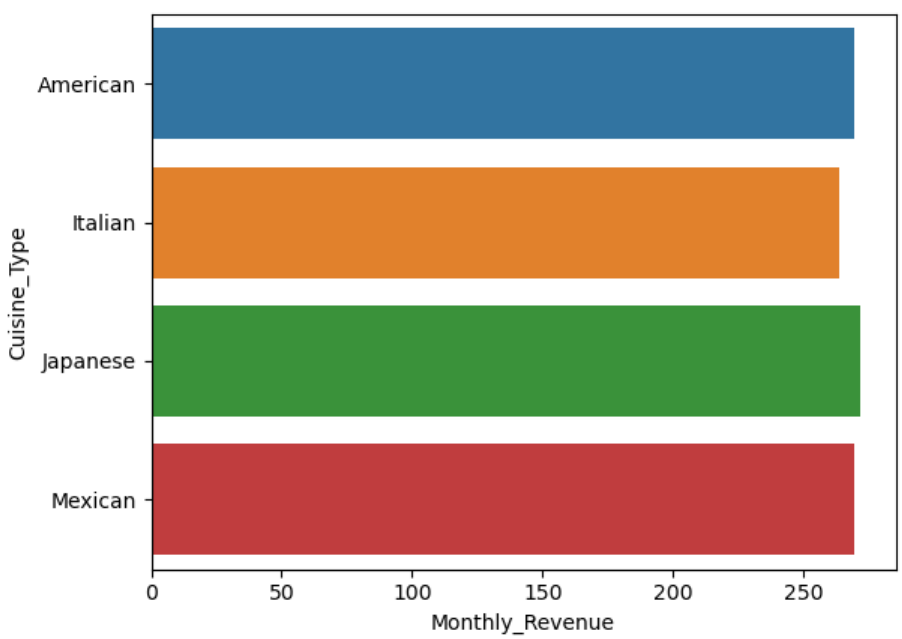




Step 4: Grouping Revenue by Categories

In [16]:
```
# group cuisine types and promotions by average monthly revenue
types = df.groupby(by='Cuisine_Type').Monthly_Revenue.agg(np.mean).reset_index()
proms = df.groupby(by='Promotions').Monthly_Revenue.agg(np.mean).reset_index()
```

In [17]:
```
# print a barplot of revenue by cuisine type
sns.barplot(data=types, x='Monthly_Revenue', y='Cuisine_Type')
plt.show()
```

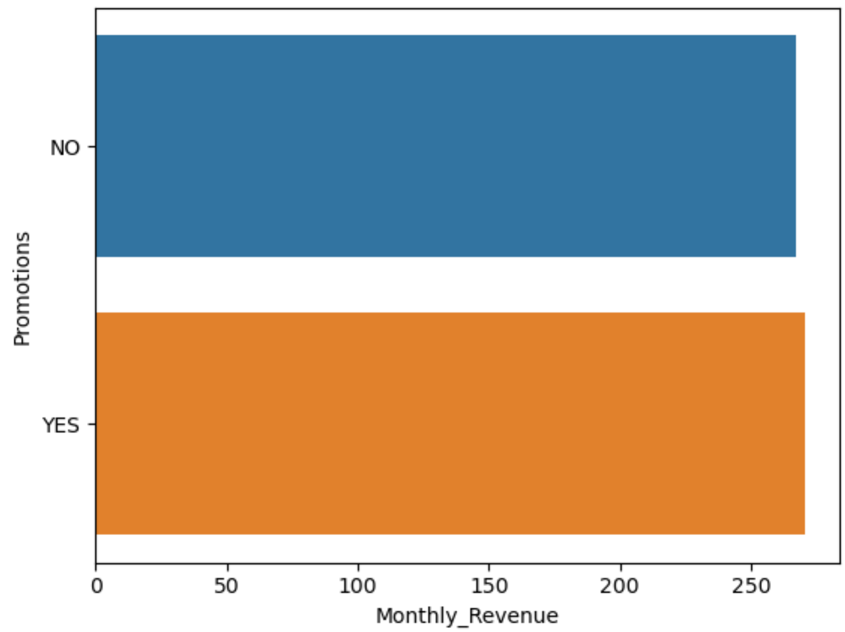


In [18]:
```
types
```

Out[18]:

	Cuisine_Type	Monthly_Revenue
0	American	269.604825
1	Italian	263.699862
2	Japanese	271.700186
3	Mexican	269.366082

In [19]:
```
# rename the 0's to NO and the 1's to YES
proms.iloc[0,0] = 'NO'
proms.iloc[1,0] = 'YES'
# print a barplot of revenue by whether or not there was a promotion
sns.barplot(data=proms, x='Monthly_Revenue', y='Promotions')
plt.show()
```



In [20]:
```
proms
```

Out[20]:

	Promotions	Monthly_Revenue
0	NO	267.188084
1	YES	270.278805

In [21]:
```
# Percent difference in aggregated monthly revenue for the presence of a promotional campaign
# to establish the campaign's worth

print('%diff in aggregated monthly restaurant revenue with ' +
      'the presence of a promotional campaign: ',
      ((proms['Monthly_Revenue'][1] / proms['Monthly_Revenue'][0]) - 1) * 100)
```

```
%diff in aggregated monthly restaurant revenue with the presence of a promotional campaign:  1.15
67584291771071
```

Step 5: Modeling

In [22]:
```
# get the dummy variables, while also dropping one of them
dum = pd.get_dummies(df, columns=['Cuisine_Type'], drop_first=True)
# set the features to be all of the variables except monthly revenue
features = dum.drop(columns='Monthly_Revenue')
# split the data into training and testing sets at an 80/20 ratio
xtrain, xtest, ytrain, ytest = train_test_split(features,
                                                dum['Monthly_Revenue'], test_size=0.2, random_stat
```

In [32]:
```python
# Create a custom wrapper for statsmodels OLS
class OLSWrapper(BaseEstimator, RegressorMixin):
    def __init__(self):
        self.model = None

    def fit(self, X, y):
        # Add a constant to the input features to account for the intercept
        X = sm.add_constant(X)
        self.model = sm.OLS(y, X).fit()
        print(self.model.summary())
        return self

    def predict(self, X):
        # Add a constant for prediction
        X = sm.add_constant(X)
        return self.model.predict(X)

    def score(self, X, y):
        predictions = self.predict(X)
        # Return R-squared score
        return 1 - np.sum((y - predictions) ** 2) / np.sum((y - np.mean(y)) ** 2)

# Create a pipeline with a custom wrapper
pipe = Pipeline([
    ('scale', StandardScaler()),
    ('model', OLSWrapper())
])

# Set up a search space
search_space = [{'model': [OLSWrapper()]},
               {'model': [Ridge(max_iter=10000)],
                'model__alpha': np.logspace(-4, 4, 50)}]

# Perform grid search with cross-validation
grid_search = GridSearchCV(pipe, search_space, cv=5, scoring='r2')
gs = grid_search.fit(xtrain, ytrain)

be = gs.best_estimator_
print(f'Best estimator: {be}')
# make a prediction using the test data
pred = be.predict(xtest)
# get R^2 and RMSE
r2 = r2_score(ytest, pred)
rmse = np.sqrt(mean_squared_error(ytest, pred))
print(f"R^2: {r2}")
print(f"RMSE: {rmse}")
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:        Monthly_Revenue   R-squared:                       0.680
Model:                            OLS   Adj. R-squared:                  0.675
Method:                 Least Squares   F-statistic:                     148.7
Date:                Fri, 26 Jul 2024   Prob (F-statistic):          1.83e-149
Time:                        02:22:11   Log-Likelihood:                -3505.1
No. Observations:                 640   AIC:                             7030.
Df Residuals:                     630   BIC:                             7075.
Df Model:                           9                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        268.9217      2.305    116.692      0.000     264.396     273.447
x1            75.8256      2.316     32.734      0.000      71.277      80.374
x2            23.7732      2.320     10.249      0.000      19.218      28.328
x3            26.8290      2.318     11.574      0.000      22.277      31.381
x4            -1.6173      2.317     -0.698      0.485      -6.167       2.932
x5            -4.1567      2.319     -1.793      0.074      -8.710       0.397
x6            -1.0656      2.316     -0.460      0.646      -5.614       3.483
x7             0.2211      2.752      0.080      0.936      -5.183       5.625
x8            -1.9680      2.783     -0.707      0.480      -7.432       3.496
x9             0.9550      2.782      0.343      0.731      -4.507       6.417
==============================================================================
Omnibus:                        2.109   Durbin-Watson:                   1.858
Prob(Omnibus):                  0.348   Jarque-Bera (JB):                2.068
Skew:                          -0.139   Prob(JB):                        0.356
Kurtosis:                       2.995   Cond. No.                         1.98
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results                            
==============================================================================
Dep. Variable:        Monthly_Revenue   R-squared:                       0.681
Model:                            OLS   Adj. R-squared:                  0.676
Method:                 Least Squares   F-statistic:                     149.3
Date:                Fri, 26 Jul 2024   Prob (F-statistic):          7.29e-150
Time:                        02:22:11   Log-Likelihood:                -3497.4
No. Observations:                 640   AIC:                             7015.
Df Residuals:                     630   BIC:                             7059.
Df Model:                           9                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        268.7920      2.277    118.048      0.000     264.321     273.263
x1            75.2285      2.288     32.878      0.000      70.735      79.722
x2            26.1852      2.296     11.403      0.000      21.676      30.695
x3            25.7532      2.291     11.241      0.000      21.254      30.252
x4             0.4491      2.281      0.197      0.844      -4.030       4.928
x5            -3.6046      2.297     -1.569      0.117      -8.116       0.907
x6             1.5748      2.293      0.687      0.492      -2.928       6.078
x7            -0.1051      2.846     -0.037      0.971      -5.695       5.485
x8            -2.0513      2.888     -0.710      0.478      -7.722       3.619
x9            -0.9865      2.875     -0.343      0.732      -6.633       4.660
==============================================================================
Omnibus:                        0.811   Durbin-Watson:                   1.889
Prob(Omnibus):                  0.667   Jarque-Bera (JB):                0.877
Skew:                          -0.013   Prob(JB):                        0.645
Kurtosis:                       2.821   Cond. No.                         2.16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results                            
==============================================================================
Dep. Variable:        Monthly_Revenue   R-squared:                       0.692
Model:                            OLS   Adj. R-squared:                  0.687
Method:                 Least Squares   F-statistic:                     156.9
Date:                Fri, 26 Jul 2024   Prob (F-statistic):          1.76e-154
Time:                        02:22:11   Log-Likelihood:                -3501.0
No. Observations:                 640   AIC:                             7022.
```

```
Df Residuals:                     630   BIC:                             7067.
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        272.0739      2.290    118.819      0.000     267.577     276.571
x1            75.8445      2.301     32.956      0.000      71.325      80.364
x2            25.5013      2.300     11.086      0.000      20.984      30.018
x3            27.4173      2.312     11.861      0.000      22.878      31.957
x4            -0.8585      2.298     -0.374      0.709      -5.371       3.654
x5            -3.6514      2.302     -1.586      0.113      -8.171       0.869
x6            -1.0937      2.301     -0.475      0.635      -5.612       3.424
x7            -0.9335      2.765     -0.338      0.736      -6.364       4.497
x8            -2.6698      2.808     -0.951      0.342      -8.184       2.844
x9            -0.9220      2.802     -0.329      0.742      -6.425       4.581
==============================================================================
Omnibus:                        2.153   Durbin-Watson:                   1.816
Prob(Omnibus):                  0.341   Jarque-Bera (JB):                2.218
Skew:                          -0.137   Prob(JB):                        0.330
Kurtosis:                       2.909   Cond. No.                         2.04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:        Monthly_Revenue   R-squared:                       0.673
Model:                            OLS   Adj. R-squared:                  0.669
Method:                 Least Squares   F-statistic:                     144.2
Date:                Fri, 26 Jul 2024   Prob (F-statistic):          1.17e-146
Time:                        02:22:11   Log-Likelihood:                -3511.7
No. Observations:                 640   AIC:                             7043.
Df Residuals:                     630   BIC:                             7088.
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        270.4416      2.329    116.139      0.000     265.869     275.014
x1            74.6859      2.338     31.950      0.000      70.095      79.276
x2            24.5162      2.336     10.496      0.000      19.930      29.103
x3            26.7485      2.350     11.383      0.000      22.134      31.363
x4             0.4820      2.344      0.206      0.837      -4.122       5.086
x5            -3.7136      2.353     -1.578      0.115      -8.334       0.907
x6            -2.4734      2.344     -1.055      0.292      -7.076       2.129
x7             2.4353      2.819      0.864      0.388      -3.101       7.972
x8             1.5313      2.854      0.537      0.592      -4.073       7.136
x9             1.4608      2.862      0.510      0.610      -4.160       7.082
==============================================================================
Omnibus:                        0.354   Durbin-Watson:                   1.849
Prob(Omnibus):                  0.838   Jarque-Bera (JB):                0.388
Skew:                          -0.056   Prob(JB):                        0.824
Kurtosis:                       2.957   Cond. No.                         2.05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:        Monthly_Revenue   R-squared:                       0.685
Model:                            OLS   Adj. R-squared:                  0.681
Method:                 Least Squares   F-statistic:                     152.5
Date:                Fri, 26 Jul 2024   Prob (F-statistic):          8.41e-152
Time:                        02:22:11   Log-Likelihood:                -3501.1
No. Observations:                 640   AIC:                             7022.
Df Residuals:                     630   BIC:                             7067.
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        269.2980      2.290    117.580      0.000     264.800     273.796
x1            75.1271      2.309     32.531      0.000      70.592      79.662
x2            23.8531      2.306     10.342      0.000      19.324      28.382
```

```
x3            24.6782      2.316     10.656      0.000      20.130      29.226
x4            -1.0789      2.303     -0.468      0.640      -5.601       3.444
x5            -3.1992      2.319     -1.379      0.168      -7.754       1.355
x6            -3.7074      2.302     -1.611      0.108      -8.228       0.813
x7             0.8092      2.792      0.290      0.772      -4.675       6.293
x8             1.6568      2.814      0.589      0.556      -3.870       7.184
x9             0.9453      2.796      0.338      0.735      -4.545       6.435
==============================================================================
Omnibus:                        0.884   Durbin-Watson:                   1.863
Prob(Omnibus):                  0.643   Jarque-Bera (JB):                0.883
Skew:                          -0.090   Prob(JB):                        0.643
Kurtosis:                       2.976   Cond. No.                         2.06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Best estimator: Pipeline(steps=[('scale', StandardScaler()),
                ('model', Ridge(alpha=7.9060432109076855, max_iter=10000))])
R^2: 0.6937014067272136
RMSE: 60.856889064989275
```

In [23]:

```python
# create a pipeline for a scale and model
pipe = Pipeline([('scale', StandardScaler()), ('model', LinearRegression())])
```

In [24]:

```python
# set up a search space for linear regression and ridge regression
search_space = [{'model': [LinearRegression()]},
                {'model': [Ridge(max_iter=10000)],
                 'model__alpha': np.logspace(-4, 4, 50)}]
```

In [25]:

```python
# make a grid search cross-validation with 5-folds over the pipe and search space
grid_search = GridSearchCV(pipe, search_space, cv=5, n_jobs=-1, verbose=0, scoring='neg_mean_squar
```

In [26]:

```python
# this function takes in training and testing variables, fits
# them to the grid search model, and prints out various results
def run_tests(x, xt, y, yt):
    # fit the training data to the grid search
    gs = grid_search.fit(x, y)
    # get the best estimator
    be = gs.best_estimator_
    print(f'Best estimator: {be}')
    # make a prediction using the test data
    pred = be.predict(xt)
    # get R^2 and RMSE
    r2 = r2_score(yt, pred)
    rmse = np.sqrt(mean_squared_error(yt, pred))
    print(f"RMSE: {rmse}")
    print(f"R^2: {r2}")
    # print out the list of variables with their respective coefficients
    coefs = be.named_steps['model'].coef_
    print("\nFeature importance:")
    for feature, coef in zip(features, coefs):
        print(f"{feature}: {coef:.4f}")
```

In [27]:

```python
run_tests(xtrain, xtest, ytrain, ytest)
```

```
Best estimator: Pipeline(steps=[('scale', StandardScaler()),
                ('model', Ridge(alpha=7.9060432109076855, max_iter=10000))])
RMSE: 60.856889064989275
R^2: 0.6937014067272136

Feature importance:
Number_of_Customers: 74.5995
Menu_Price: 24.5487
Marketing_Spend: 26.0052
Average_Customer_Spending: -0.5611
Promotions: -3.5897
Reviews: -1.3689
Cuisine_Type_Italian: 0.4578
Cuisine_Type_Japanese: -0.6830
Cuisine_Type_Mexican: 0.2322
```

In [28]:
```python
# this function runs a tree regression model on the data using a parameter grid
def run_tree_regression():
    # set the initial state of the regressor
    tree_reg = DecisionTreeRegressor(random_state=1)
    # define a grid of potential parameters
    param_grid = {
        'max_depth': [None, 10, 20, 30, 40, 50],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    # create a grid search using 5 cross-folds on the tree regression model
    grid_search = GridSearchCV(estimator=tree_reg, param_grid=param_grid,
                               cv=5, scoring='neg_mean_squared_error', n_jobs=-1, verbose=0)
    # apply the grid search to the training data
    grid_search.fit(xtrain, ytrain)
    # determine the best estimator
    be = grid_search.best_estimator_
    # use the best model to make predictions on the test data
    preds = be.predict(xtest)
    RMSE = np.sqrt(mean_squared_error(ytest, preds))
    R_sqaured = r2_score(ytest, preds)
    # print out the results of the best model's performance
    print(f'Best estimator: {be}')
    print(f'RMSE: {RMSE}')
    print(f'R^2: {R_sqaured}')
```

In [29]:
```python
run_tree_regression()
```

```
Best estimator: DecisionTreeRegressor(max_depth=10, min_samples_leaf=4, min_samples_split=10,
                      random_state=1)
RMSE: 77.33858179980125
R^2: 0.5053273330434154
```

In [33]:
```python
# this function creates an ensemble model of ridge, tree, random forest and gradient boosting
# that applies a stacking regressor to the training data
def run_ensemble():
    # initialize each of the regression models
    ridge_reg = Ridge()
    tree_reg = DecisionTreeRegressor(random_state=1)
    rf_reg = RandomForestRegressor(random_state=1)
    gb_reg = GradientBoostingRegressor(random_state=1)
    # define the estimators for each model
    estimators = [
        ('ridge', ridge_reg),
        ('tree', tree_reg),
        ('rf', rf_reg),
        ('gb', gb_reg)
    ]
    # create a stack of models with a 5-fold cross-validation
    stacking_reg = StackingRegressor(
        estimators=estimators,
        cv=5
    )
    # fit the stack to the training data
    stacking_reg.fit(xtrain, ytrain)
    # use the fitted model to make predictions on the test data
    ypred_train = stacking_reg.predict(xtrain)
    ypred_test = stacking_reg.predict(xtest)
    # gather the assessment metrics
    rmse = np.sqrt(mean_squared_error(ytest, ypred_test))
    rsquared = r2_score(ytest, ypred_test)
    # print out the metrics
    print(f'RMSE: {rmse}')
    print(f'R^2: {rsquared}')
```

In [34]:
```python
run_ensemble()
```

```
RMSE: 61.785734154105945
R^2: 0.6842801198816617
```