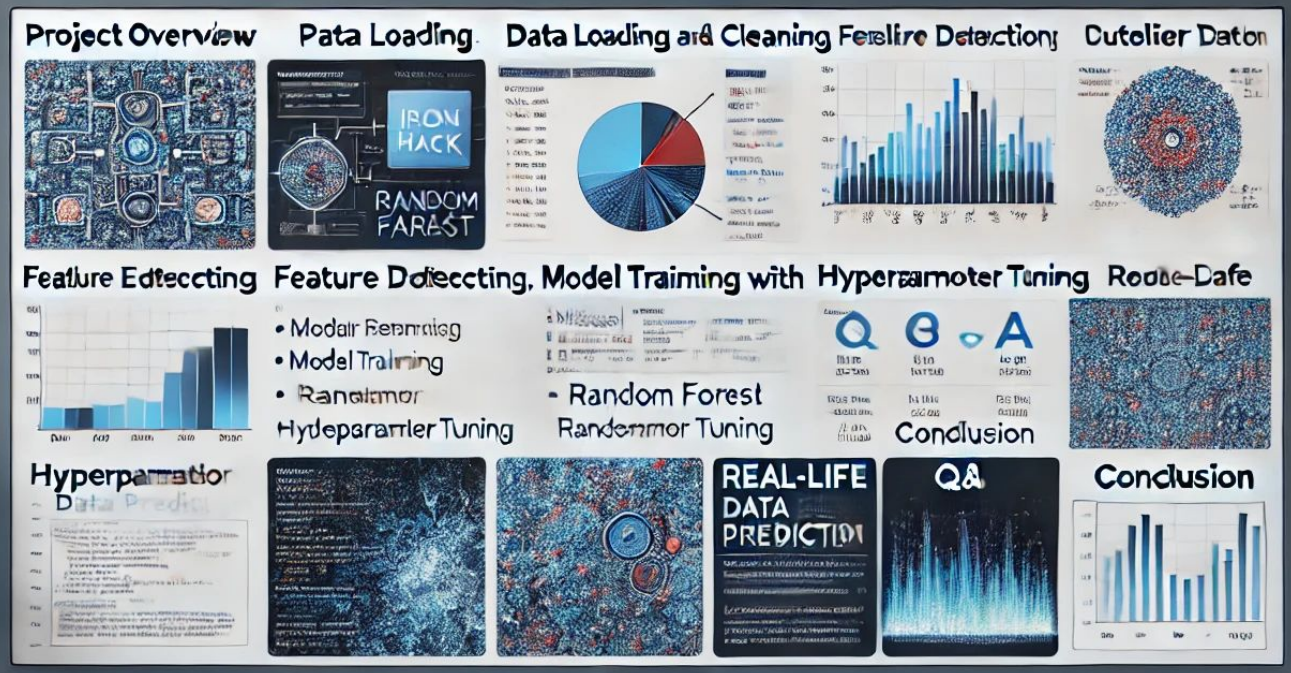
Machine Learning Project



Data Loading and Cleaning

```python
# Load the necessary libraries and dataset
import pandas as pd
file_path = r'C:\Users\mktmi\Documents\ironhack\AI_Engineering\Work\Week2\Labs\IronKaggle\sales.csv'
sales_data = pd.read_csv(file_path)

# Step 1: Data Cleaning

# Convert 'date' column to datetime
sales_data['date'] = pd.to_datetime(sales_data['date'])

# One-hot encoding of 'state_holiday'
sales_data_encoded = pd.get_dummies(sales_data, columns=['state_holiday'], drop_first=True)

# Drop 'Unnamed: 0' column
sales_data_cleaned = sales_data_encoded.drop(columns=['Unnamed: 0'])
```

Feature Engineering

```
# Step 2: Feature Engineering

# Extract additional features from the 'date' column
sales_data_cleaned['year'] = sales_data_cleaned['date'].dt.year
sales_data_cleaned['month'] = sales_data_cleaned['date'].dt.month
sales_data_cleaned['day'] = sales_data_cleaned['date'].dt.day
sales_data_cleaned['week_of_year'] = sales_data_cleaned['date'].dt.isocalendar().week
```

Additional Preprocessing/ Outliers

```
#Additional Preprocessing Steps

# Step 1: Extract additional features from the 'date' column
sales_data_cleaned['year'] = sales_data_cleaned['date'].dt.year
sales_data_cleaned['month'] = sales_data_cleaned['date'].dt.month
sales_data_cleaned['day'] = sales_data_cleaned['date'].dt.day
sales_data_cleaned['week_of_year'] = sales_data_cleaned['date'].dt.isocalendar().week

# Step 2: Checking for outliers

from scipy import stats

# Step 1: Calculate the Z-score for 'sales' and 'nb_customers_on_day'
sales_z_scores = stats.zscore(sales_data_cleaned['sales'])
customers_z_scores = stats.zscore(sales_data_cleaned['nb_customers_on_day'])

# Step 2: Identify outliers (Z-score > 3 or Z-score < -3)
outliers_sales = (abs(sales_z_scores) > 3)
outliers_customers = (abs(customers_z_scores) > 3)

# Step 3: Remove the outliers from the dataset
sales_data_no_z_outliers = sales_data_cleaned[~(outliers_sales | outliers_customers)]

# Display the size of the dataset after removing outliers
sales_data_no_z_outliers.info(), sales_data_no_z_outliers.head()
```

Model Training

- Split the Data

```
from sklearn.model_selection import train_test_split

# Step 1: Define features (X) and target (y)
X = sales_data_no_z_outliers.drop(columns=['sales', 'date'])
y = sales_data_no_z_outliers['sales']

# Step 2: Perform an 80/20 train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Display the size of the training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

✓ 0.1s

```
((503917, 13), (125980, 13), (503917,), (125980,))
```

Model Training with Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Initialize the Linear Regression model
linear_model = LinearRegression()

# Step 2: Train the model on the training data
linear_model.fit(X_train, y_train)

# Step 3: Make predictions on the test set
y_pred = linear_model.predict(X_test)

# Step 4: Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

mse, r2
```

2] ✓ 0.3s Python

```
(1673060.7451727523, 0.8649088693573257)
```

Model Training with Random Forest

```python
# Random Forest

from sklearn.ensemble import RandomForestRegressor

# Step 1: Initialize the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Step 2: Train the Random Forest model on the training data
rf_model.fit(X_train, y_train)

# Step 3: Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Step 4: Evaluate the model's performance
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

mse_rf, r2_rf
```

✓ 4m 13.5s Python

```
(695351.1767549412, 0.9438539354099544)
```

Hyperparameter Tuning

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Define the parameter grid for RandomizedSearchCV
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Step 2: Set up RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=RandomForestRegressor(random_state=42),
                                   param_distributions=param_grid,
                                   n_iter=10,  # Number of combinations to try
                                   cv=3,       # 3-fold cross-validation
                                   random_state=42, n_jobs=-1)

# Fit the model
random_search.fit(X_train, y_train)
```

Hyperparameter Tuning

```python
# Step 3: Retrieve the best hyperparameters and evaluate performance
best_rf_model = random_search.best_estimator_  # Now this works after fitting
y_pred_rf_tuned = best_rf_model.predict(X_test)

# Step 4: Calculate MSE and R²
mse_rf_tuned = mean_squared_error(y_test, y_pred_rf_tuned)
r2_rf_tuned = r2_score(y_test, y_pred_rf_tuned)

# Display the best parameters and model performance
print("Best Parameters: ", random_search.best_params_)
print(f"MSE: {mse_rf_tuned}, R²: {r2_rf_tuned}")
```

```
Best Parameters:  {'n_estimators': 200, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 30}
MSE: 1305008.891728689, R²: 0.8946271812359279
```

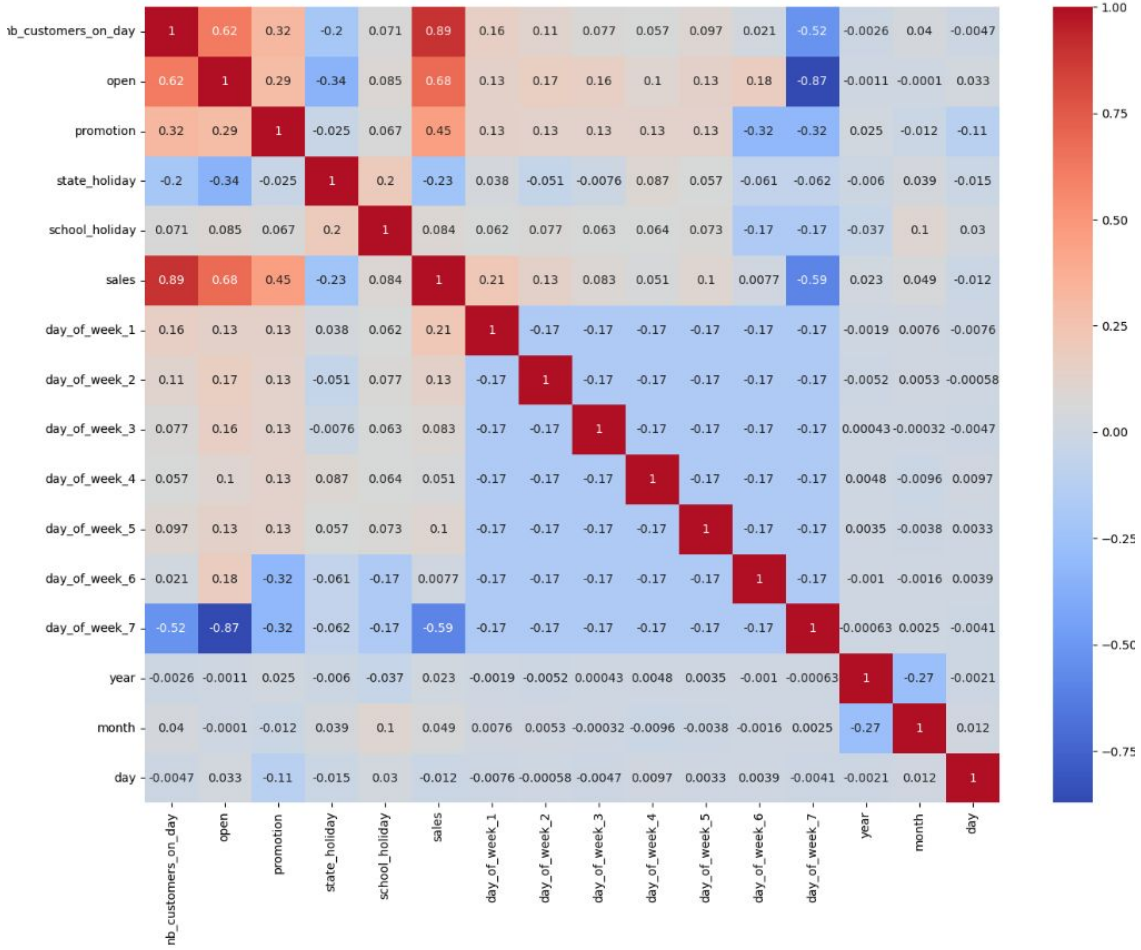
Real-Life Data Prediction

```
# Step 1: Load the real-life dataset
real_life_data = pd.read_csv('REAL_DATA.csv')

# Step 2: Preprocess the data
real_life_data = real_life_data.drop('index', axis=1)
real_life_data['date'] = pd.to_datetime(real_life_data['date'], dayfirst=True)
real_life_data = pd.get_dummies(real_life_data, columns=['state_holiday'], drop_first=True)
real_life_data['year'] = real_life_data['date'].dt.year
real_life_data['month'] = real_life_data['date'].dt.month
real_life_data['day'] = real_life_data['date'].dt.day
real_life_data['week_of_year'] = real_life_data['date'].dt.isocalendar().week

# Step 3: Drop unnecessary columns and predict sales
X_real = real_life_data.drop(columns=['date'])
predicted_sales = rf_model.predict(X_real)

# Step 4: Add predictions to the dataset
rd = pd.read_csv('REAL_DATA.csv')
rd['sales'] = predicted_sales
# Step 5: Save the dataset with predictions to a CSV file
rd.to_csv('predicted_sales_real_data.csv', index=False)
print("Predictions saved to 'predicted_sales_real_data.csv'")
```

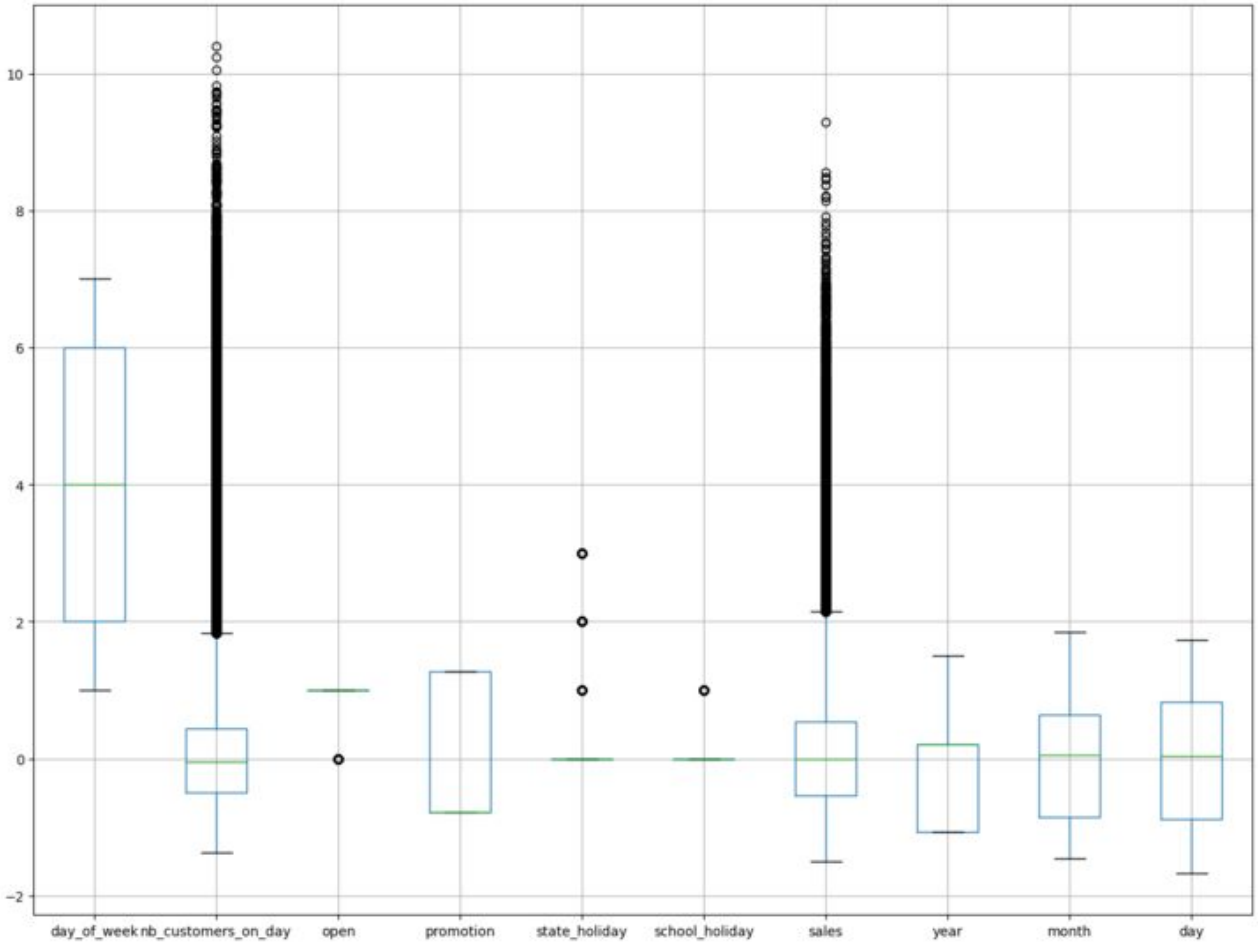
Lesson Learned (not that we need them!!! ;)

- We dropped StoreID column
- We removed outliers from nb_customers_on_day only and not on target sales
- We standardized the data: Helpful for the linear regression but useless for RandomForrest
- With all this, we reached the following r2 scores: 0.85 on LinearRegression, 0.86 on RandomForrest and 0.88 on XBoost

Lesson Learned (not that we need them!!! ;)



Lesson Learned (not that we need them!!! ;)



Conclusion and Next Steps

- Successfully built and optimized a predictive model for shop sales.
- Generated predictions for real-life data using the Random Forest model.
- Next steps: Explore additional models and fine-tune for specific shop categories.

Questions
and Feedback