ANOMALY DETECTION IN TIME SERIES DATA OF NEW YORK CITY TAXI DEMAND

By: JUJJAVARAPU SUJAN CHOWDARY

DECEMBER 2023

INTRODUCTION

Finding anomalies in the New York City Taxi Demand dataset is the goal of this project. The NYC Taxi and Limousine Commission is the source of the dataset, which offers a detailed account of all taxi passengers combined into 30-minute blocks. The dataset is notable for having anomalies associated with significant occasions like the NYC Marathon, Thanksgiving, Christmas, New Year's Day, and a snowfall. An essential component of data analysis is anomaly detection, which makes it possible to spot odd trends or occurrences that greatly depart from the norm. Understanding and spotting abnormalities in the context of taxi demand is crucial for maximizing resource allocation and service scheduling.

In order to find anomalies in the taxi demand dataset, this project uses a machine learning technique called the Isolation Forest algorithm. The Isolation Forest works well in situations where anomalies are distinct and uncommon, and it is especially useful for finding outliers in large datasets. The report will examine each step of the project, including feature engineering, data exploration, preprocessing, training the model, and anomaly visualization. By the project's conclusion, we hope to shed light on the temporal trends in New York City's taxi demand and draw attention to unusual occurrences that have a big influence on passenger volumes.

OBJECTIVE

This project's main goal is to use machine learning techniques for anomaly detection on the New York City Taxi Demand dataset. Among the precise aims and objectives are:

1. **Anomaly Identification:** To find and isolate anomalies in the taxi demand dataset, apply the Isolation Forest algorithm.

2. **Temporal Pattern Analysis:** To learn how anomalies correspond with significant occasions like the NYC Marathon, Thanksgiving, Christmas, New Year's Day, and a snowstorm, investigate and analyze temporal patterns in taxi demand.

3. **Feature Engineering:** Apply the required feature engineering methods to improve the model's capacity to detect and classify anomalies.

4. **Visualization:** To enable a more intuitive understanding, produce interactive visualizations that highlight the detected anomalies and offer insights into their temporal distribution.

ABOUT ANOMALY DETECTION IN TIME SERIES DATA

Finding irregularities, deviations, or unexpected patterns in a series of observations over time is largely dependent on anomaly detection in time series data. Since temporal dependencies are frequently present in time series data, using specialized techniques is crucial for effective anomaly detection. For the purposes of this project, anomalies in taxi passenger counts are instances of unusual activity, and we concentrate on the New York City Taxi Demand dataset.

- **Temporal Dynamics:** Anomalies can appear as abrupt spikes, drops, or asymmetric patterns in the temporal sequence. Time series data shows how a phenomenon evolves over time.
- **Types of Anomalies:** Time series data can contain point anomalies, which are significantly different individual data points, contextual anomalies, which are deviations taking into account the surrounding context, and collective anomalies, which are abnormally behaved groups of data points.
- **Difficulties with Time Series Deviation Identification:**
  - **Trends and Seasonality:** Seasonal trends and patterns can make it difficult to identify anomalies.
  - **Noise and Variability:** It may be difficult to distinguish between typical and abnormal behaviour in time series data due to inherent noise and variability.

Methods for Identifying Time Series Anomalies:

- **Statistical Methods:** Based on departures from expected values, identify anomalies using statistical measures like mean, median, and standard deviation.

- **Machine Learning Algorithms:** For more advanced anomaly detection, use machine learning algorithms such as LSTM (Long Short-Term Memory) networks, One-Class SVM, and Isolation Forest.
- **Forecasting Models:** Use time series forecasting models to forecast expected values and identify deviations as anomalies, such as ARIMA and Prophet.
- **Hybrid Approaches:** Utilize the advantages of both statistical and machine learning techniques, combine several approaches to improve detection accuracy.

Time Series Anomaly Detection Advantages:

- **Proactive Issue Identification:** Proactive intervention and potential issue mitigation are made possible by early anomaly detection.
- **Resource Optimization:** Better planning and resource allocation based on an enhanced comprehension of anomalous patterns in temporal data.
- **Improved Decision-Making:** By pointing out significant occurrences or departures from expected behaviour, accurate anomaly detection aids in the making of well-informed decisions.

This project's use of the Isolation Forest algorithm for anomaly detection fits in with the temporal structure of the taxi demand dataset, making it a useful tool for locating and examining anomalies in relation to noteworthy New York City occurrences.

APPROACH

The New York City Taxi Demand dataset is subjected to anomaly detection using a methodical approach by the project. The sequential procedure entails:

1. **Data Loading:** Bring in the required Python libraries for data manipulation, analysis, visualization, and machine learning, such as NumPy, Pandas, Holoviews, and Scikit-learn.
2. **Dataset Exploration:** To comprehend the structure, features, and anomalies of the New York City Taxi Demand dataset ({nyc_taxi.csv}), perform a preliminary exploration of the dataset. Examine data types, null values, and summary statistics to learn more about the features of the dataset.
3. **Preprocessing the data:** Address any missing values to make sure the dataset is tidy and prepared for examination. To ensure accurate temporal analysis, convert the timestamp column to a datetime format. Examine and prepare elements necessary for anomaly detection, such as resampling data for various time frames.
4. **Feature Engineering:** To improve the dataset for more accurate anomaly detection, add new features like weekday, hour, day, month, and year.
   To capture trends and temporal dependencies, create rolling averages and lag features.
5. **Exploratory Data Analysis (EDA):** To visualize the overall distribution of taxi demand, find outliers, and comprehend temporal patterns, use Holoviews for exploratory data analysis.
6. **Isolation Forest Model Training:** Use Scikit-learn's Isolation Forest algorithm to find anomalies.
   Use the engineered and preprocessed features to train the model.

7. **Anomaly Visualization:** To visualize the anomalies that have been found on top of the overall demand curve, use Holoviews. This will help you understand exceptional events intuitively.
8. **Assessment of Performance:** Evaluate the anomaly detection model's performance using relevant metrics, like F1-score, precision, and recall. Adjust model hyperparameters as needed to maximize efficiency.

To efficiently identify anomalies in the New York City Taxi Demand dataset, a combination of data exploration, preprocessing, feature engineering, and machine learning techniques is employed in the selected approach. A comprehensive grasp of the temporal patterns and exceptional events in taxi demand is provided by the incorporation of interactive visualizations made possible by Holoviews, which improves the interpretability of the results.

METHODOLOGY

1. Data Loading and Exploration:

- Import necessary libraries: NumPy, Pandas, Holoviews, Scikit-learn, and others.
- Load the New York City Taxi Demand dataset (`nyc_taxi.csv`) and examine its structure and features.
- Check for data types, null values, and conduct summary statistics to understand the dataset's characteristics.

	timestamp	value
0	2014-07-01 00:00:00	10844
1	2014-07-01 00:30:00	8127
2	2014-07-01 01:00:00	6210
3	2014-07-01 01:30:00	4656
4	2014-07-01 02:00:00	3820
...	...	...
10315	2015-01-31 21:30:00	24670
10316	2015-01-31 22:00:00	25721
10317	2015-01-31 22:30:00	27309
10318	2015-01-31 23:00:00	26591
10319	2015-01-31 23:30:00	26288

10320 rows × 2 columns

2. Data Preprocessing:

- Address missing values in the dataset to ensure data cleanliness.
- Convert the timestamp column to a datetime format for temporal analysis.
- Explore and preprocess features relevant to anomaly detection, including resampling data for different time intervals.

```
Null Count:
 timestamp    0
value        0
dtype: int64

Data Types:
 timestamp    datetime64[ns]
value                 int64
dtype: object

Date Range:

Start:   2014-07-01 00:00:00
End:     2015-01-31 23:30:00
Days:    214 days 23:30:00
```

3. Feature Engineering:

- Create additional features to enhance the dataset for anomaly detection, such as weekday, hour, day, month, and year.

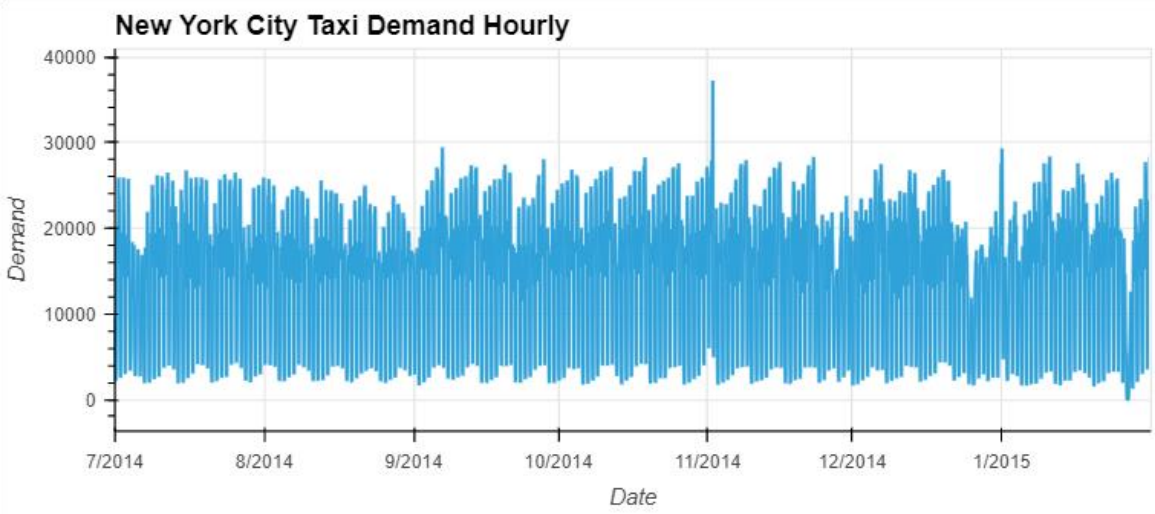
- Generate lag features and rolling averages to capture temporal dependencies and trends in taxi demand.

	timestamp	value	Weekday	Hour	Day	Month	Year	Month_day	Lag	Rolling_Mean
1	2014-07-02 00:00:00	15,284	Wednesday	0	2	7	2014	2	15540.979167	15412.572917
2	2014-07-03 00:00:00	14,795	Thursday	0	3	7	2014	3	15284.166667	15206.590278
3	2014-07-04 00:00:00	11,512	Friday	0	4	7	2014	4	14794.625000	14282.885417
4	2014-07-05 00:00:00	11,572	Saturday	0	5	7	2014	5	11511.770833	13740.766667
5	2014-07-06 00:00:00	11,464	Sunday	0	6	7	2014	6	11572.291667	13361.350694

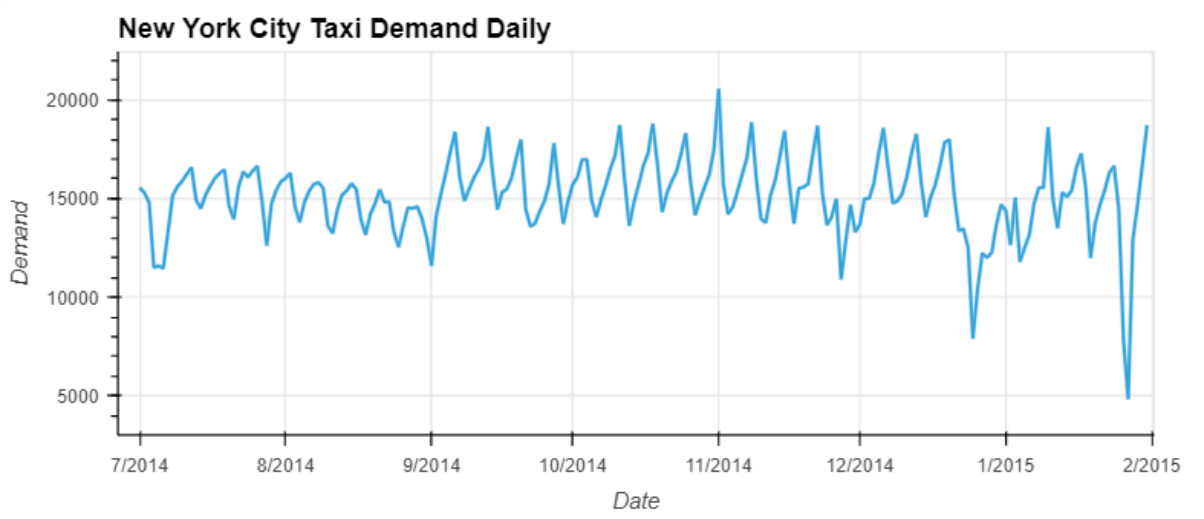
4. Exploratory Data Analysis (EDA) with Holoviews:

- Utilize Holoviews to visually explore the distribution of taxi demand, identify outliers, and understand temporal patterns.
- Create interactive plots for hourly, daily, and weekly demand, facilitating a deeper understanding of temporal dynamics.

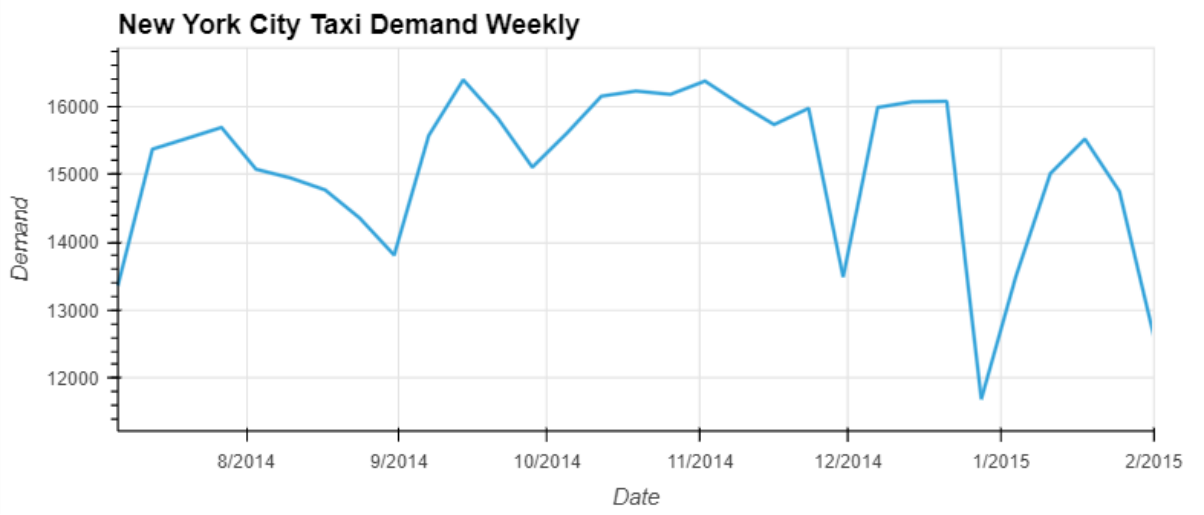
NEW YORK CITY TAXI DEMAND HOURLY:



NEW YORK CITY TAXI DEMAND DAILY:



NEW YORK CITY TAXI DEMAND WEEKLY:



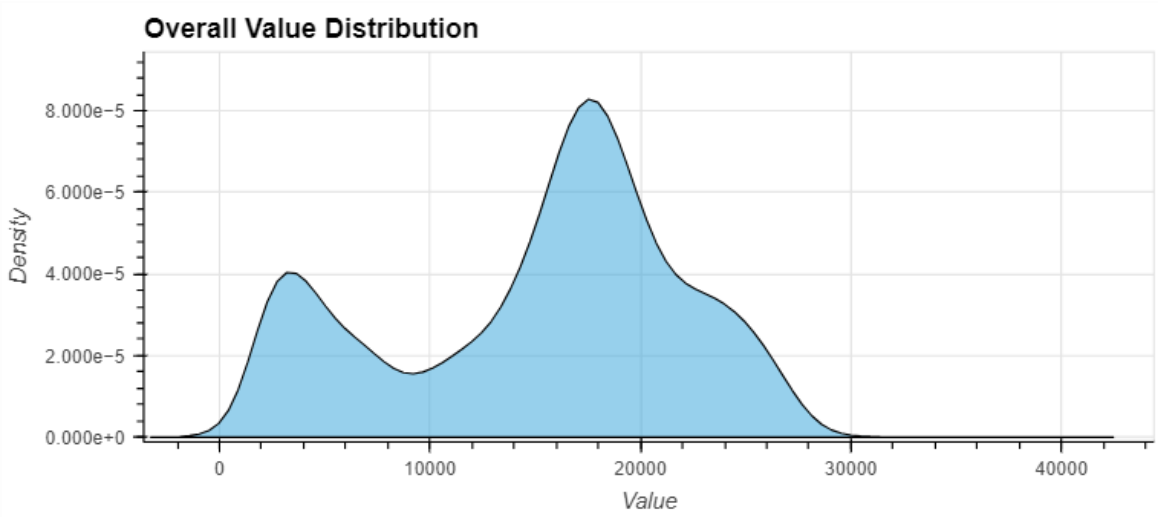
5. Isolation Forest Model Training:

- Implement the Isolation Forest algorithm from Scikit-learn for anomaly detection.
- Split the dataset into training and testing sets.
- Train the Isolation Forest model on the training set using relevant features.

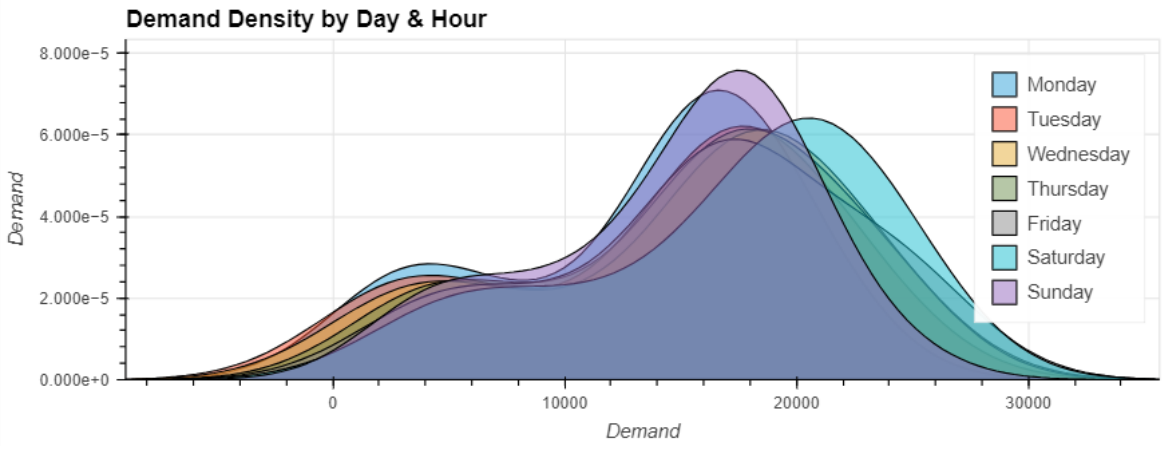
6. Anomaly Visualization with Holoviews:

- Utilize Holoviews to visualize the identified anomalies on top of the overall demand curve.
- Create interactive plots highlighting anomalies during major events such as the NYC marathon, Thanksgiving, Christmas, New Year's Day, and a snowstorm.

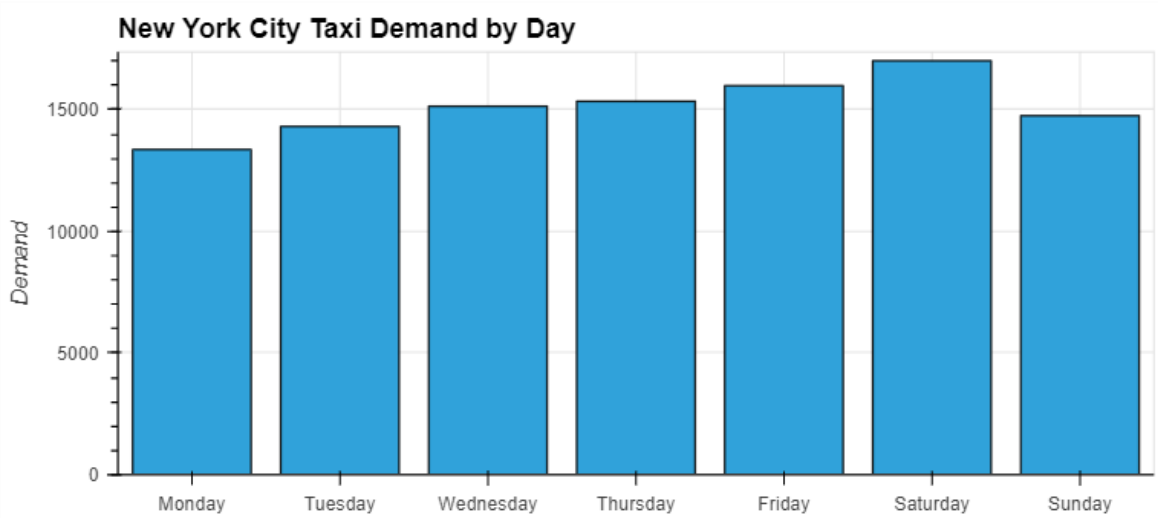
OVERALL VALUE DISTRIBUTION



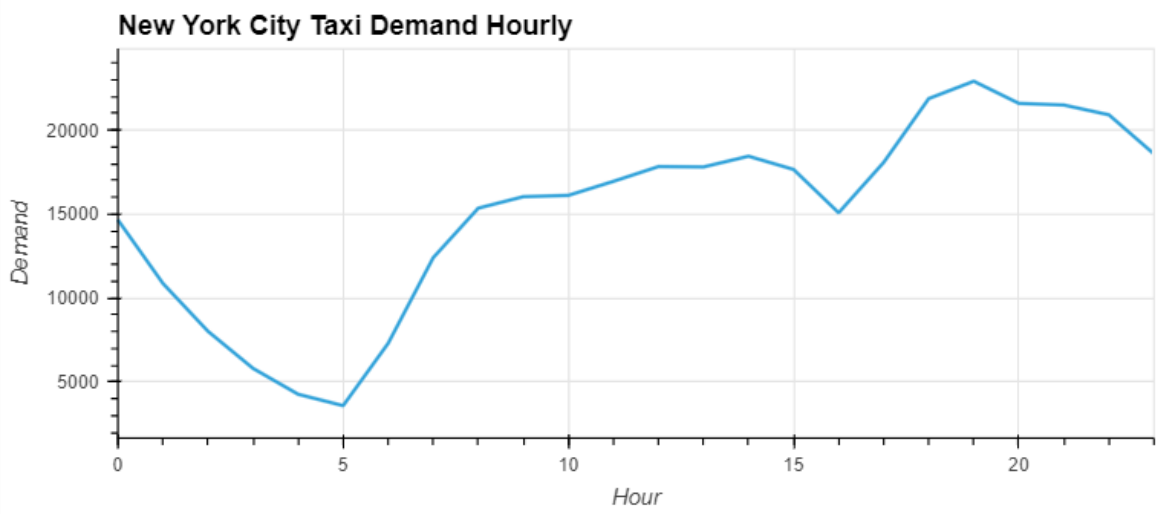
DEMAND DENSITY BY DAY AND HOUR



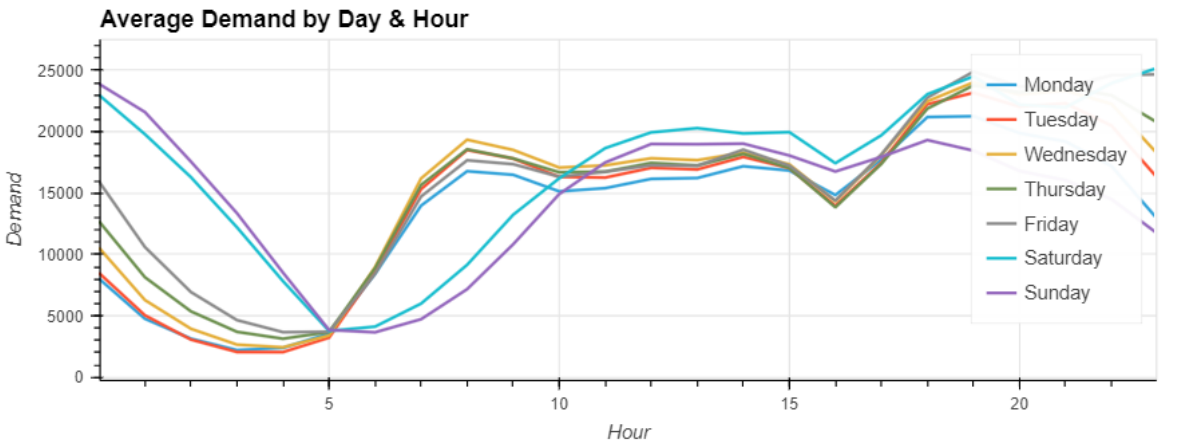
NEW YORK CITY TAXI DEMAND BY DAY:



NEW YORK CITY TAXI DEMAND HOURLY:



AVERAGE DEMAND BY DAY AND HOUR:



In [34]:
```
df_hourly.tail()
```

Out[34]:

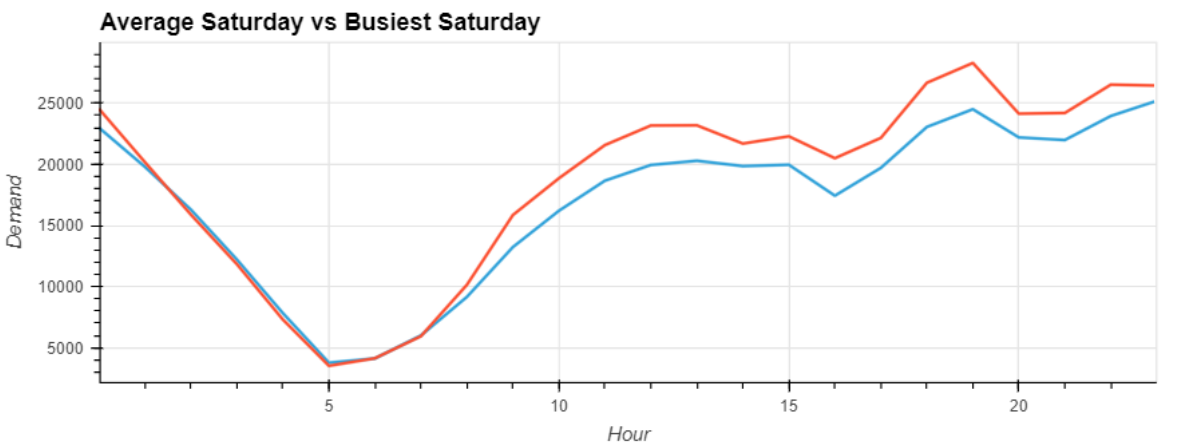
	timestamp	value	Weekday	Hour	Day	Month	Year	Month_day	Lag	Rolling_Mean	value_Average	Outliers	Score	value_Average
5155	2015-01-31 19:00:00	28288.5	Saturday	19	5	1	2015	31	26665.0	23537.214286	24501.870968	0.0	0.001273	24501.870968
5156	2015-01-31 20:00:00	24138.0	Saturday	20	5	1	2015	31	28288.5	23673.571429	22193.758065	0.0	0.004279	22193.758065
5157	2015-01-31 21:00:00	24194.5	Saturday	21	5	1	2015	31	24138.0	24031.214286	21983.241935	0.0	0.046274	21983.241935
5158	2015-01-31 22:00:00	26515.0	Saturday	22	5	1	2015	31	24194.5	24635.714286	23949.951613	1.0	0.027190	23949.951613
5159	2015-01-31 23:00:00	26439.5	Saturday	23	5	1	2015	31	26515.0	25485.071429	25192.516129	NaN	-0.004814	25192.516129

In [35]:
```
df_daily.tail()
```

Out[35]:

	timestamp	value	Weekday	Hour	Day	Month	Year	Month_day	Lag	Rolling_Mean	value_Average	value_Average
210	2015-01-27	4834.541667	Tuesday	0	1	1	2015	27	7818.979167	12874.401786	14308.778898	14308.778898
211	2015-01-28	12947.562500	Wednesday	0	2	1	2015	28	4834.541667	12628.976190	15137.875672	15137.875672
212	2015-01-29	14686.145833	Thursday	0	3	1	2015	29	12947.562500	12526.473214	15343.679435	15343.679435
213	2015-01-30	16676.625000	Friday	0	4	1	2015	30	14686.145833	12580.431548	15983.970430	15983.970430
214	2015-01-31	18702.479167	Saturday	0	5	1	2015	31	16676.625000	12875.732143	17007.264785	17007.264785

AVERAGE SATURDAY VS BUSIEST SATURDAY:



7. **Insight Generation:**

- Derive insights from the detected anomalies, examining how they align with major events and understanding their impact on taxi demand.

The methodology encompasses a systematic process, combining data exploration, preprocessing, feature engineering, machine learning model training, and interactive visualization to effectively detect and understand anomalies in the New York City Taxi Demand dataset. The iterative nature of the methodology allows for refinement based on insights gained during the project.

RESULT

Anomaly Visualization with Holoviews:

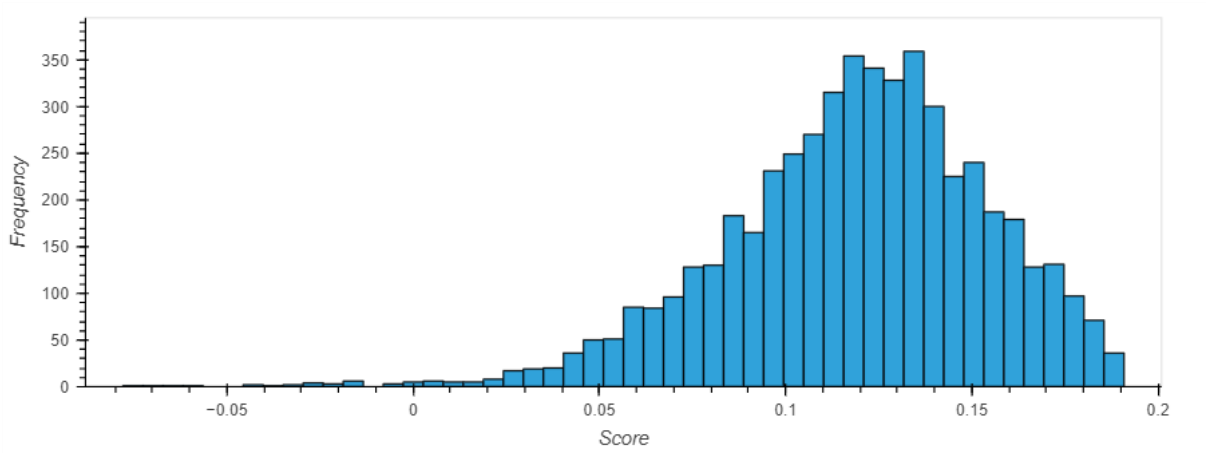
- The resulting interactive plot is displayed. It includes points for anomalies with tooltips and a curve representing overall taxi demand. Users can interact with the plot using tools such as box select, lasso select, and tap.

The count of data points identified as anomalies in the `df_hourly` DataFrame based on the condition "Outliers == 1." The length of the resulting DataFrame indicates the number of occurrences identified as outliers.

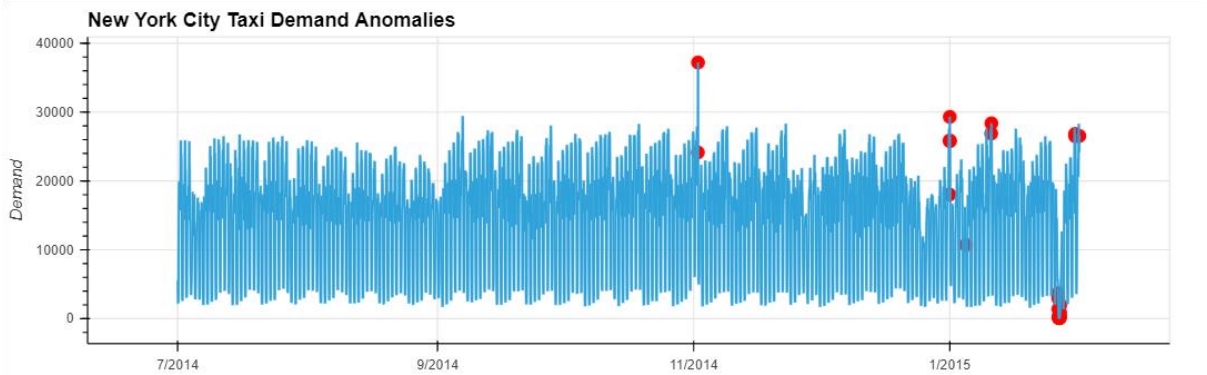
>>> 26

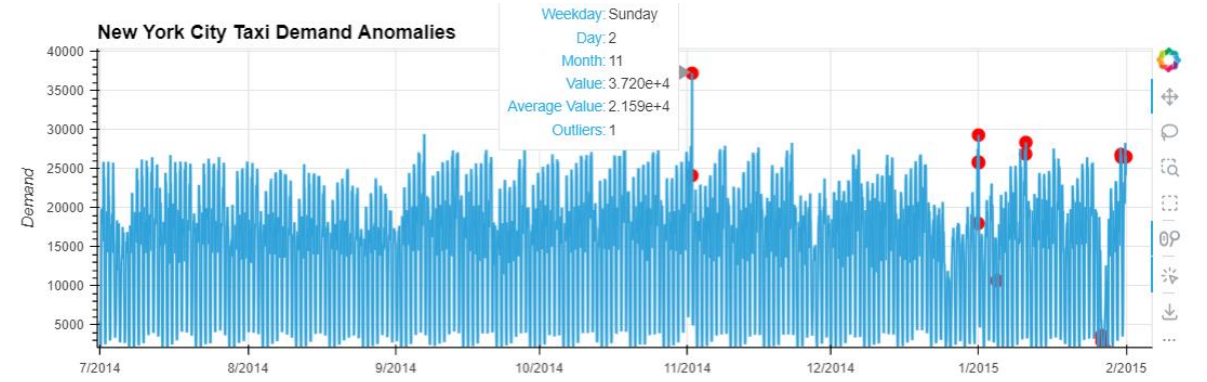
Anomaly scores calculated by the Isolation Forest model:

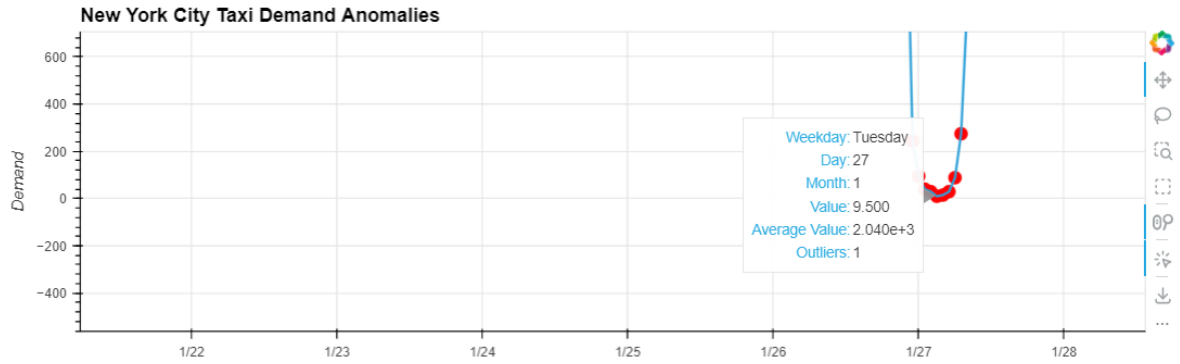
This plot provides a visual representation of the distribution of anomaly scores, helping to understand the spread and concentration of scores across different bins. The x-axis represents the anomaly scores, and the y-axis represents the frequency of data points within each score range.



This plot helps visually identify and explore anomalies in taxi demand, providing additional information on weekdays, days, months, values, and average values for each anomalous data point.







So the anomalous part of the data is on upper is 2nd Nov, Sunday and lower is 27th Jan, Tuesday.

CODE REPOSITORY

```
# Import necessary libraries
import numpy as np       # NumPy for numerical operations
import pandas as pd      # Pandas for data manipulation
import os                # OS module for interacting with the operating system
import matplotlib.dates as mdates  # Matplotlib dates module for handling dates
import holoviews as hv   # HoloViews for interactive visualizations
from holoviews import opts  # HoloViews options for customization
hv.extension('bokeh')    # Enable the Bokeh backend for HoloViews
from bokeh.models import HoverTool  # Bokeh HoverTool for interactive tooltips
from IPython.display import HTML, display  # IPython display for rendering HTML content

# Ignore warnings to keep the output clean
import warnings
warnings.filterwarnings("ignore")

# Import IsolationForest from scikit-learn
from sklearn.ensemble import IsolationForest
```

```
df =
pd.read_csv(r"C:\Users\chowd\Downloads\archive\realKnownCause\realKnownCause\nyc_taxi.csv", parse_dates=['timestamp'])


(df.head(5)
 .style
 .set_caption('New York City Taxi Demand')
 .format({'value':"{:,.0f}"})
)

def overview(df: pd.DataFrame, timestamp_col: str = None) -> None:
    print('Null Count:\n', df.isnull().sum(),'\n')
    print('Data Types:\n', df.dtypes)

    if timestamp_col is not None:
        print('\nDate Range:\n\nStart:\t',df[timestamp_col].min())
        print('End:\t',df[timestamp_col].max())
        print('Days:\t',(df[timestamp_col].max() - df[timestamp_col].min()))

overview(df, timestamp_col='timestamp')

Hourly = hv.Curve(df.set_index('timestamp').resample('H').mean()).opts(
    opts.Curve(title="New York City Taxi Demand Hourly", xlabel="Date", ylabel="Demand",
               width=700, height=300,tools=['hover'],show_grid=True))
Hourly.opts(shared_axes=False)

Daily = hv.Curve(df.set_index('timestamp').resample('D').mean()).opts(
    opts.Curve(title="New York City Taxi Demand Daily", xlabel="Date", ylabel="Demand",
               width=700, height=300,tools=['hover'],show_grid=True))

Daily.opts(shared_axes=False)

Weekly = hv.Curve(df.set_index('timestamp').resample('W').mean()).opts(
```

```
   opts.Curve(title="New York City Taxi Demand Weekly", xlabel="Date",
ylabel="Demand",
            width=700, height=300,tools=['hover'],show_grid=True))
Weekly.opts(shared_axes=False)

# A variety of resamples which I may or may not use
df_hourly = df.set_index('timestamp').resample('H').mean().reset_index()
df_daily = df.set_index('timestamp').resample('D').mean().reset_index()
df_weekly = df.set_index('timestamp').resample('W').mean().reset_index()
print(df_hourly.head(5))
print(df_daily.head(5))
print(df_weekly.head(5))

for DataFrame in [df_hourly, df_daily]:
   DataFrame['Weekday'] = (pd.Categorical(DataFrame['timestamp'].dt.strftime('%A'),
                                 categories=['Monday', 'Tuesday', 'Wednesday', 'Thursday','Friday',
'Saturday', 'Sunday'])
                        )
   DataFrame['Hour'] = DataFrame['timestamp'].dt.hour
   DataFrame['Day'] = DataFrame['timestamp'].dt.weekday
   DataFrame['Month'] = DataFrame['timestamp'].dt.month
   DataFrame['Year'] = DataFrame['timestamp'].dt.year
   DataFrame['Month_day'] = DataFrame['timestamp'].dt.day
   DataFrame['Lag'] = DataFrame['value'].shift(1)
   DataFrame['Rolling_Mean'] = DataFrame['value'].rolling(7, min_periods=1).mean()
   DataFrame = DataFrame.dropna()

   (DataFrame.head(5)
    .style
    .format({'value':"{:,.0f}"})
   )

(hv.Distribution(df['value'])
```

```
.opts(opts.Distribution(title="Overall Value Distribution",
                        xlabel="Value",
                        ylabel="Density",
                        width=700, height=300,
                        tools=['hover'],show_grid=True)
    ))


by_weekday = df_hourly.groupby(['Hour','Weekday']).mean()['value'].unstack()

plot = hv.Distribution(by_weekday['Monday'], label='Monday') *
hv.Distribution(by_weekday['Tuesday'], label='Tuesday') *
hv.Distribution(by_weekday['Wednesday'], label='Wednesday') *
hv.Distribution(by_weekday['Thursday'], label='Thursday') *
hv.Distribution(by_weekday['Friday'], label='Friday') *
hv.Distribution(by_weekday['Saturday'], label='Saturday')
*hv.Distribution(by_weekday['Sunday'],
label='Sunday').opts(opts.Distribution(title="Demand Density by Day & Hour"))

plot.opts(opts.Distribution(width=800, height=300,tools=['hover'],show_grid=True,
ylabel="Demand", xlabel="Demand"))


hv.Bars(df_hourly[['value','Weekday']].groupby('Weekday').mean()).opts(
    opts.Bars(title="New York City Taxi Demand by Day", xlabel="", ylabel="Demand",
              width=700, height=300,tools=['hover'],show_grid=True))


hv.Curve(df_hourly[['value','Hour']].groupby('Hour').mean()).opts(
    opts.Curve(title="New York City Taxi Demand Hourly", xlabel="Hour",
ylabel="Demand",
              width=700, height=300,tools=['hover'],show_grid=True))


by_weekday = df_hourly.groupby(['Hour','Weekday']).mean()['value'].unstack()

plot = hv.Curve(by_weekday['Monday'], label='Monday') *
hv.Curve(by_weekday['Tuesday'], label='Tuesday') * hv.Curve(by_weekday['Wednesday'],
label='Wednesday') * hv.Curve(by_weekday['Thursday'], label='Thursday') *
hv.Curve(by_weekday['Friday'], label='Friday') * hv.Curve(by_weekday['Saturday'],
label='Saturday') *hv.Curve(by_weekday['Sunday'],
label='Sunday').opts(opts.Curve(title="Average Demand by Day & Hour"))

plot.opts(opts.Curve(width=800, height=300,tools=['hover'],show_grid=True,
ylabel="Demand"))
```

```
df_hourly.groupby(['Hour','Weekday']).mean()['value'].unstack().plot()

df_hourly = (df_hourly
             .join(df_hourly.groupby(['Hour','Weekday'])['value'].mean(),
                   on = ['Hour', 'Weekday'], rsuffix='_Average')
            )

df_daily = (df_daily
            .join(df_daily.groupby(['Hour','Weekday'])['value'].mean(),
                  on = ['Hour', 'Weekday'], rsuffix='_Average')
           )

df_hourly.tail()
df_daily.tail()

sat_max = (df_hourly
           .query("Day == 5")
           .set_index('timestamp')
           .loc['2015-01-31':'2015-01-31']
           .reset_index()['value']
          )



avg_sat = (df_hourly
           .groupby(['Weekday','Hour'])['value']
           .mean()
           .unstack()
           .T['Saturday']
          )

avg_max_comparison = hv.Curve(avg_sat, label='Average Saturday') * hv.Curve(sat_max,
label='Busiest Saturday').opts(opts.Curve(title="Average Saturday vs Busiest Saturday"))
```

```
avg_max_comparison.opts(opts.Curve(width=800,
height=300,tools=['hover'],show_grid=True, ylabel="Demand", show_legend=False))


#Clear nulls

df_hourly.dropna(inplace=True)


# Daily

df_daily_model_data = df_daily[['value', 'Hour', 'Day',
'Month','Month_day','Rolling_Mean']].dropna()


# Hourly

model_data = df_hourly[['value', 'Hour', 'Day', 'Month_day', 'Month','Rolling_Mean','Lag',
'timestamp']].set_index('timestamp').dropna()

model_data


def run_isolation_forest(model_data: pd.DataFrame, contamination=0.005,
n_estimators=200, max_samples=0.7) -> pd.DataFrame:


    IF = (IsolationForest(random_state=0,

                          contamination=contamination,

                          n_estimators=n_estimators,

                          max_samples=max_samples)

          )


    IF.fit(model_data)


    output = pd.Series(IF.predict(model_data)).apply(lambda x: 1 if x == -1 else 0)


    score = IF.decision_function(model_data)


    return output, score


outliers, score = run_isolation_forest(model_data)
```

```
df_hourly = (df_hourly
            .assign(Outliers = outliers)
            .assign(Score = score)
           )

df_hourly

IF = IsolationForest(random_state=0, contamination=0.005, n_estimators=200,
max_samples=0.7)
IF.fit(model_data)

# New Outliers Column
df_hourly['Outliers'] = pd.Series(IF.predict(model_data)).apply(lambda x: 1 if x == -1 else 0)

# Get Anomaly Score
score = IF.decision_function(model_data)

# New Anomaly Score column
df_hourly['Score'] = score

df_hourly.head()

def outliers(thresh):
    print(f'Number of Outliers below Anomaly Score Threshold {thresh}:')
    print(len(df_hourly.query(f"Outliers == 1 & Score <= {thresh}")))

tooltips = [
    ('Weekday', '@Weekday'),
    ('Day', '@Month_day'),
    ('Month', '@Month'),
    ('Value', '@value'),
    ('Average Value', '@value_Average'),
```

```
    ('Outliers', '@Outliers')
]
hover = HoverTool(tooltips=tooltips)

hv.Points(df_hourly.query("Outliers == 1")).opts(size=10, color='#ff0000') *
hv.Curve(df_hourly).opts(opts.Curve(title="New York City Taxi Demand Anomalies",
xlabel="", ylabel="Demand" , height=300, responsive=True,tools=[hover,'box_select',
'lasso_select', 'tap'],show_grid=True))

len(df_hourly.query("Outliers == 1"))

frequencies, edges = np.histogram(score, 50)
hv.Histogram((edges, frequencies)).opts(width=800, height=300,tools=['hover'],
xlabel='Score')

# Function to view number of outliers at a given threshold
outliers(0.05)

for num in (np.arange(-0.08, 0.2, 0.02)):
    print(len(df_hourly.query(f"Outliers == 1 & Score <= {num}")))
    num_outliers = len(df_hourly.query(f"Outliers == 1 & Score <= {num}"))

hover = HoverTool(tooltips=tooltips)

hv.Points(df_hourly.query("Outliers == 1 & Score <= 0.05")).opts(size=10, color='#ff0000') *
hv.Curve(df_hourly).opts(opts.Curve(title="New York City Taxi Demand", xlabel="",
ylabel="Demand" , height=300, responsive=True,tools=[hover,'box_select', 'lasso_select',
'tap'],show_grid=True))

import numpy as np

# List of threshold values to evaluate
threshold_values = np.arange(0.01, 0.2, 0.01)
```

```
for threshold in threshold_values:
    # Convert anomaly scores to binary labels based on the threshold
    y_pred_binary = (df_hourly['Score'] <= threshold).astype(int)

    # Calculate binary classification metrics
    precision = precision_score(df_hourly['Outliers'], y_pred_binary)
    recall = recall_score(df_hourly['Outliers'], y_pred_binary)
    f1 = f1_score(df_hourly['Outliers'], y_pred_binary)
    roc_auc = roc_auc_score(df_hourly['Outliers'], y_pred_binary)
    pr_auc = average_precision_score(df_hourly['Outliers'], y_pred_binary)

    # Print metrics for each threshold
    print(f'Threshold: {threshold:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1-Score: {f1:.4f}')
    print(f'ROC-AUC: {roc_auc:.4f}')
    print(f'PR-AUC: {pr_auc:.4f}')
    print('-' * 40)

import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, roc_curve



y_true = df_hourly['Outliers']
y_scores = df_hourly['Score']

precision, recall, thresholds_pr = precision_recall_curve(y_true, y_scores)
fpr, tpr, thresholds_roc = roc_curve(y_true, y_scores)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
```

```
plt.plot(recall, precision, label='Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(fpr, tpr, label='ROC Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve')
plt.legend()

plt.tight_layout()
plt.show()
```


- **NAME:** JUJJAVARAPU SUJAN CHOWDARY
- **EMAIL:** chowdarysujan27@gmail.com
- **Phone:** 7780539873