Background

Prior to beginning this project, I knew nothing about blockchain nor Ethereum specifically. I had heard market or dove deeper into how it works other than just knowing that it is out there. I am taking the op more about blockchain and Ethereum cryptocurreny. Looking at the data and, along the way, learn wh good avenue for a deep learning opportunity with the topic. Throughout this project, I plan on examini from the month of October in both 2018 and 2019. Comparing these attributes within each month and analysis opportunities for me to understand exactly how each attribute relates to the other and how a another. I believe the information discovered during this project will not only be helpful for me to unde but also for anyone else who may be interested in learning more about the topics. Not only having an relationships between these attributes proves valuable to investors and miners of Ethereum crypto-cu the month and actions on certain holidays and how each influences the Ethereum crypto-currency ma be able to look ahead in the market.

Data

The data I will be examining in this project comes from the Google Bigquery dataset called crypto_eth For the purposes of understanding the blockchain and the Ethereum cryptocurrency, I will focus on th contracts, and transactions table. I wish I could use data from the other 4 tables, but they take a very tables I chose. I may add on a week in October examination of the 4 tables for which querying the wh purposes.

Within the blocks table, I chose the attributes of the date, the block number, the miner activity, the diff difficulty experenced on the block up to the time of the transaction in question, the size of the block, t the transaction, and the transaction count.

Within the contracts table, I chose the attributes of the date and block number that the contract was c whether or not the contract was an erc20 contract or an erc721 contract.

And lastly within the transaction table, I chose the attributes of the date and the block number the tra reciever address of the transaction, the value, the gas price at the time of the transaction, whether a n transaction, and the status of the transaction (either a 1 for a success or 0 for failure)

Setting up Big Query Connection

```
from google.colab import auth
auth.authenticate_user()
```

```
print('Authenticated')
```

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
import pandas as pd
import datetime as dt
import plotly.express as px
import plotly.graph_objects as go
import os
import seaborn as sns
%matplotlib inline
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import pydotplus

project_id = 'data-470-project'

from google.cloud import bigquery

client = bigquery.Client(project=project_id)
```

Final Plan Outline

Blocks Table

Query Google BigQuery

```
blocks_18 = client.query('''
  SELECT DATE(timestamp) as date, number, miner, difficulty, total_difficulty, size, gas_li
  FROM `bigquery-public-data.crypto_ethereum.blocks`
  WHERE DATE(timestamp) >= "2018-10-01" AND DATE(timestamp) < "2018-10-31" ''').to_datafram

blocks_19 = client.query('''
  SELECT DATE(timestamp) as date, number, miner, difficulty, total_difficulty, size, gas_li
  FROM `bigquery-public-data.crypto_ethereum.blocks`
  WHERE DATE(timestamp) >= "2019-10-01" AND DATE(timestamp) < "2019-10-31" ''').to_datafram
```

Concat serparte dfs into one df called blocks and add in dayofyear, year, and day of week columns

```
frames = [blocks_18, blocks_19]
blocks = pd.concat(frames)
blocks_num = blocks.groupby(by='date', as_index=False, sort = 'True').agg({'number': pd.Ser
blocks_min = blocks.groupby(by='date', as_index=False, sort = 'True').agg({'miner': pd.Seri
blocks_min = blocks_min[['miner']]
blocks_num['date'] = pd.to_datetime(blocks_num['date'], errors='coerce')
blocks_num['dayofyear'] = blocks_num['date'].dt.dayofyear
blocks_num['year'] = pd.DatetimeIndex(blocks_num['date']).year
blocks_num['dow'] = blocks_num['date'].dt.day_name()


blocks_diff = blocks.groupby(by='date', as_index=False, sort = 'True').agg({'difficulty': p
blocks_diff = blocks_diff[['difficulty']]


blocks_gas = blocks.groupby(by='date', as_index=False, sort = 'True').agg({'gas_used': pd.S
blocks_gas = blocks_gas[['gas_used']]


blocks_trans = blocks.groupby(by='date', as_index=False, sort = 'True').agg({'transaction_c
blocks_trans = blocks_trans[['transaction_count']]
```

Concat all calculations from above

```
blocks_comp = pd.concat([blocks_num, blocks_min, blocks_diff, blocks_gas, blocks_trans], ax


blocks_comp.head()
```

Plots

```
blocks_comp['year'] = blocks_comp['year'].astype(str)
fig = px.scatter_matrix(blocks_comp, dimensions=["number", "miner", "difficulty", "gas_used
fig.show()


fig = px.line(blocks_comp, x="dayofyear", y="number", color = "year", hover_name= "dow", ti
fig.update_layout(
    xaxis=dict(
```

```
            title_text = "Day Of Year",
            showline=True,
            showgrid=False,
            showticklabels=True,
            linecolor='rgb(204, 204, 204)',
            linewidth=2,
            ticks='outside',
            tickfont=dict(
                family='Arial',
                size=12,
                color='rgb(82, 82, 82)',
            ),
        ),
        yaxis=dict(
            title_text = "Block Count",
            showgrid=False,
            zeroline=False,
            showline=False,
            showticklabels=True,
        ),
        margin=dict(
            autoexpand=False,
            l=100,
            r=100,
            t=110,
        ),
        barmode = 'group',
        legend_orientation = "h",
        showlegend=True,
        plot_bgcolor='white'
    )
    fig.show()
```

```
fig = px.line(blocks_comp, x="dayofyear", y="miner", color = "year", hover_name= "dow", tit
fig.update_layout(
    xaxis=dict(
        title_text = "Day Of Year",
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        title_text = "Miner Count",
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=True,
    ),
    margin=dict(
        autoexpand=False,
        l=100,
        r=100,
        t=110,
    ),
    barmode = 'group',
    legend_orientation = "h",
    showlegend=True,
    plot_bgcolor='white'
)
fig.show()
```

```
fig = px.line(blocks_comp, x="dayofyear", y="difficulty", color = "year", hover_name= "dow"
fig.update_layout(
    xaxis=dict(
        title_text = "Day Of Year",
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        title_text = "Average Difficulty",
        showgrid=False,
        zeroline=False,
```

```
            showline=False,
            showticklabels=True,
        ),
        margin=dict(
            autoexpand=False,
            l=100,
            r=100,
            t=110,
        ),
        barmode = 'group',
        legend_orientation = "h",
        showlegend=True,
        plot_bgcolor='white'
    )
    fig.show()
```



```
    fig = px.line(blocks_comp, x="dayofyear", y="gas_used", color = "year", hover_name= "dow",
    fig.update_layout(
        yaxis=dict(
```

```
        xaxis=dict(
            title_text = "Day Of Year",
            showline=True,
            showgrid=False,
            showticklabels=True,
            linecolor='rgb(204, 204, 204)',
            linewidth=2,
            ticks='outside',
            tickfont=dict(
                family='Arial',
                size=12,
                color='rgb(82, 82, 82)',
            ),
        ),
        yaxis=dict(
            title_text = "Total Gas Used",
            showgrid=False,
            zeroline=False,
            showline=False,
            showticklabels=True,
        ),
        margin=dict(
            autoexpand=False,
            l=100,
            r=100,
            t=110,
        ),
        barmode = 'group',
        legend_orientation = "h",
        showlegend=True,
        plot_bgcolor='white'
    )
    fig.show()
```

```
fig = px.line(blocks_comp, x="dayofyear", y="transaction_count", color = "year", hover_name
fig.update_layout(
    xaxis=dict(
        title_text = "Day Of Year",
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        title_text = "Transaction Count",
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=True,
    ),
    margin=dict(
        autoexpand=False,
        l=100,
        r=100,
        t=110,
    ),
    barmode = 'group',
    legend_orientation = "h",
    showlegend=True,
    plot_bgcolor='white'
)
fig.show()
```

Random Forrest: Can we predict the day of the week?

```
labels = np.array(blocks_comp['dow'])
labels[:5]
```

```
feature = blocks_comp.drop('date', axis = 1)
feature_1 = feature.drop('dayofyear', axis = 1)
features = feature_1.drop('year', axis = 1)
features = features.drop('dow', axis = 1)

features.head()
```

```
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42)
rf.fit(features, labels)
```

```
predictions = rf.predict(features)
blocks_comp['predicted'] = predictions
blocks_comp.head()
```

```
blocks_comp.count()
```

```
blocks_comp[['dow']].count()
```

```
blocks_comp.groupby('dow').count()[['predicted']]
```

```
train_features, test_features, train_labels, test_labels = \
train_test_split(features, labels, test_size = 0.25, random_state = 42)

print("Size of the training data", train_features.shape)
print("Size of the test data", test_features.shape)
print("Size of the training labels", train_labels.shape)
print("Size of the test labels", test_labels.shape)
```

```
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42)

rf.fit(train_features, train_labels);

predictions = rf.predict(test_features)

print(predictions)
print(test_labels)
```

```
conf_mat = pd.crosstab(predictions, test_labels, rownames=['Actual DOW'], \
            colnames=['Predicted DOW'])
conf_mat
```

```
rf.feature_importances_
```

```
list(zip(train_features, rf.feature_importances_))
```

```
feature_list = list(features.columns.values)
feature_list
```

```
unique_pos = blocks_comp['dow'].unique()
unique_pos
target_names = list(unique_pos)
target_names
```

```
estimator = rf.estimators_[0]
from sklearn.tree import export_graphviz
# Export as dot file
tree = export_graphviz(estimator, out_file='tree.dot',
```

```
                    feature_names = feature_list,
                    class_names = target_names,
                    rounded = True, proportion = False,
                    precision = 2, filled = True)

##
# Convert to png using system command (requires Graphviz)
from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])

# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'tree.png')
```

I'm not too sure why this is coming out so large

Contracts

Query Google BigQuery contracts table

```
contracts_18 = client.query('''
SELECT DATE(block_timestamp) as date, block_number, address, is_erc20, is_erc721
FROM `bigquery-public-data.crypto_ethereum.contracts`
WHERE DATE(block_timestamp) >= "2018-10-01" AND DATE(block_timestamp) < "2018-10-31"''').to

contracts_19 = client.query('''
SELECT DATE(block_timestamp) as date, block_number, address, is_erc20, is_erc721
FROM `bigquery-public-data.crypto_ethereum.contracts`
WHERE DATE(block_timestamp) >= "2019-10-01" AND DATE(block_timestamp) < "2019-10-31"''').to


frames = [contracts_18, contracts_19]
contracts = pd.concat(frames)
contracts_num = contracts.groupby(by='date', as_index=False, sort = 'True').agg({'block_num
contracts_20 = contracts.groupby(by='date', as_index=False, sort = 'True').agg({'is_erc20':
contracts_721 = contracts.groupby(by='date', as_index=False, sort = 'True').agg({'is_erc721
contracts_num['date'] = pd.to_datetime(contracts_num['date'], errors='coerce')
contracts_num['dayofyear'] = contracts_num['date'].dt.dayofyear
contracts_num['year'] = pd.DatetimeIndex(contracts_num['date']).year
contracts_num['dow'] = contracts_num['date'].dt.day_name()
contracts_20 = contracts_20[['is_erc20']]
contracts_721 = contracts_721[['is_erc721']]
contracts_comp = pd.concat([contracts_num, contracts_20, contracts_721,], axis=1, sort=Fals


contracts_comp.head()
```

Plots

```
fig = px.line(contracts_comp, x="dayofyear", y="block_number", color = "year", hover_name=
fig.update_layout(
    xaxis=dict(
        title_text = "Day Of Year",
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        title_text = "Block Count",
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=True,
    ),
    margin=dict(
        autoexpand=False,
        l=100,
        r=100,
        t=110,
    ),
    barmode = 'group',
    legend_orientation = "h",
    showlegend=True,
    plot_bgcolor='white'
)
fig.show()
```

```
fig = px.line(contracts_comp, x="dayofyear", y="is_erc20", color = "year", hover_name= "dow
fig.update_layout(
    xaxis=dict(
        title_text = "Day Of Year",
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        title_text = "ERC20 Contract Count",
        showgrid=False,
        zeroline=False,
        showline=False,
```

```
            showline=False,
            showticklabels=True,
        ),
        margin=dict(
            autoexpand=False,
            l=100,
            r=100,
            t=110,
        ),
        barmode = 'group',
        legend_orientation = "h",
        showlegend=True,
        plot_bgcolor='white'
    )
    fig.show()
```



```
    fig = px.line(contracts_comp, x="dayofyear", y="is_erc721", color = "year", hover_name= "do
    fig.update_layout(
        xaxis=dict(
```

```
            title_text = "Day Of Year",
            showline=True,
            showgrid=False,
            showticklabels=True,
            linecolor='rgb(204, 204, 204)',
            linewidth=2,
            ticks='outside',
            tickfont=dict(
                family='Arial',
                size=12,
                color='rgb(82, 82, 82)',
            ),
        ),
        yaxis=dict(
            title_text = "ERC721 Contract Count",
            showgrid=False,
            zeroline=False,
            showline=False,
            showticklabels=True,
        ),
        margin=dict(
            autoexpand=False,
            l=100,
            r=100,
            t=110,
        ),
        barmode = 'group',
        legend_orientation = "h",
        showlegend=True,
        plot_bgcolor='white'
    )
    fig.show()
```

```
labels = np.array(contracts_comp['dow'])
feature = contracts_comp.drop('date', axis = 1)
feature_1 = feature.drop('dayofyear', axis = 1)
features = feature_1.drop('year', axis = 1)
features = features.drop('dow', axis = 1)
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42)
rf.fit(features, labels)
predictions = rf.predict(features)
contracts_comp['predicted'] = predictions

contracts_comp.head()
```

```
contracts_comp.count()
```

```
contracts_comp[['dow']].count()
```

```
contracts_comp.groupby('dow').count()[['predicted']]
```

```
train_features, test_features, train_labels, test_labels = \
train_test_split(features, labels, test_size = 0.25, random_state = 42)
print("Size of the training data", train_features.shape)
print("Size of the test data", test_features.shape)
print("Size of the training labels", train_labels.shape)
print("Size of the test labels", test_labels.shape)
```

```
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42)
rf.fit(train_features, train_labels)
predictions = rf.predict(test_features)
print(predictions)
print(test_labels)
```

```
conf_mat = pd.crosstab(predictions, test_labels, rownames=['Actual DOW'], \
            colnames=['Predicted DOW'])
conf_mat
```

```
rf.feature_importances_
```

```
list(zip(train_features, rf.feature_importances_))
```

```
feature_list = list(features.columns.values)
unique_pos = contracts_comp['dow'].unique()
target_names = list(unique_pos)
estimator = rf.estimators_[0]
from sklearn.tree import export_graphviz
# Export as dot file
tree = export_graphviz(estimator, out_file='tree.dot',
                feature_names = feature_list,
                class_names = target_names,
                rounded = True, proportion = False,
                precision = 2, filled = True)

##
# Convert to png using system command (requires Graphviz)
from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])

# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'tree.png')
```

Transaction

I ran into a problem here because the transactions table was so huge that it took too long to run and t
the table up and run it by week.

```
trans_18 = client.query('''
SELECT DATE(block_timestamp) as date, block_number, from_address, to_address, value, gas_pr
```

```
FROM `bigquery-public-data.crypto_ethereum.transactions`
WHERE DATE(block_timestamp) >= "2018-10-01" AND DATE(block_timestamp) < "2018-10-7"''').to_

trans_19 = client.query('''
SELECT DATE(block_timestamp) as date, block_number, from_address, to_address, value, gas_pr
FROM `bigquery-public-data.crypto_ethereum.transactions`
WHERE DATE(block_timestamp) >= "2019-10-01" AND DATE(block_timestamp) < "2019-10-7"''').to_


trans_18_2 = client.query('''
SELECT DATE(block_timestamp) as date, block_number, from_address, to_address, value, gas_pr
FROM `bigquery-public-data.crypto_ethereum.transactions`
WHERE DATE(block_timestamp) >= "2018-10-08" AND DATE(block_timestamp) < "2018-10-18"''').to

trans_19_2 = client.query('''
SELECT DATE(block_timestamp) as date, block_number, from_address, to_address, value, gas_pr
FROM `bigquery-public-data.crypto_ethereum.transactions`
WHERE DATE(block_timestamp) >= "2019-10-08" AND DATE(block_timestamp) < "2019-10-18"''').to


trans_18_3 = client.query('''
SELECT DATE(block_timestamp) as date, block_number, from_address, to_address, value, gas_pr
FROM `bigquery-public-data.crypto_ethereum.transactions`
WHERE DATE(block_timestamp) >= "2018-10-19" AND DATE(block_timestamp) < "2018-10-31"''').to

trans_19_3 = client.query('''
SELECT DATE(block_timestamp) as date, block_number, from_address, to_address, value, gas_pr
FROM `bigquery-public-data.crypto_ethereum.transactions`
WHERE DATE(block_timestamp) >= "2019-10-19" AND DATE(block_timestamp) < "2019-10-31"''').to


frames = [trans_18, trans_18_2, trans_18_3, trans_19, trans_19_2, trans_19_3]
trans = pd.concat(frames)
trans_num = trans.groupby(by='date', as_index=False, sort = 'True').agg({'block_number': pd
trans_from = trans.groupby(by='date', as_index=False, sort = 'True').agg({'from_address': p
trans_to = trans.groupby(by='date', as_index=False, sort = 'True').agg({'to_address': pd.Se
trans_value = trans.groupby(by='date', as_index=False, sort = 'True').agg({'value': pd.Seri
trans_price = trans.groupby(by='date', as_index=False, sort = 'True').agg({'gas_price': pd.
trans_status = trans.groupby(by='date', as_index=False, sort = 'True').agg({'receipt_status
trans_num['date'] = pd.to_datetime(trans_num['date'], errors='coerce')
trans_num['dayofyear'] = trans_num['date'].dt.dayofyear
trans_num['year'] = pd.DatetimeIndex(trans_num['date']).year
trans_num['dow'] = trans_num['date'].dt.day_name()
trans_from = trans_from[['from_address']]
trans_to = trans_to[['to_address']]
trans_value = trans_value[['value']]
trans_price = trans_price[['gas_price']]
trans_status = trans_status[['receipt_status']]
trans_comp = pd.concat([trans_num, trans_to, trans_from, trans_value, trans_price, trans_st


trans_comp.head()
```

```
fig = px.line(trans_comp, x="dayofyear", y="block_number", color = "year", hover_name= "dow
fig.update_layout(
    xaxis=dict(
        title_text = "Day Of Year",
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        title_text = "Block Count",
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=True,
    ),
    margin=dict(
        autoexpand=False,
        l=100,
        r=100,
        t=110,
    ),
    barmode = 'group',
    legend_orientation = "h",
    showlegend=True,
    plot_bgcolor='white'
)
fig.show()
```

```
fig = px.line(trans_comp, x="dayofyear", y="from_address", color = "year", hover_name= "dow
fig.update_layout(
    xaxis=dict(
        title_text = "Day Of Year",
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        title_text = "Sender Address Count",
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=True,
    ),
    margin_dict(
```

```
        margin=dict(
            autoexpand=False,
            l=100,
            r=100,
            t=110,
        ),
        barmode = 'group',
        legend_orientation = "h",
        showlegend=True,
        plot_bgcolor='white'
    )
    fig.show()
```



```
    fig = px.line(trans_comp, x="dayofyear", y="to_address", color = "year", hover_name= "dow",
    fig.update_layout(
        xaxis=dict(
            title_text = "Day Of Year",
            showline=True,
            showgrid=False,
```

```
            showticklabels=True,
            linecolor='rgb(204, 204, 204)',
            linewidth=2,
            ticks='outside',
            tickfont=dict(
                family='Arial',
                size=12,
                color='rgb(82, 82, 82)',
            ),
        ),
        yaxis=dict(
            title_text = "Receiver Address Count",
            showgrid=False,
            zeroline=False,
            showline=False,
            showticklabels=True,
        ),
        margin=dict(
            autoexpand=False,
            l=100,
            r=100,
            t=110,
        ),
        barmode = 'group',
        legend_orientation = "h",
        showlegend=True,
        plot_bgcolor='white'
    )
    fig.show()
```

```
fig = px.line(trans_comp, x="dayofyear", y="value", color = "year", hover_name= "dow", titl
fig.update_layout(
    xaxis=dict(
        title_text = "Day Of Year",
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        title_text = "Average Value",
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=True,
    ),
    margin=dict(
        autoexpand=False,
        l=100,
        r=100,
        t=110,
    ),
    barmode = 'group',
    legend_orientation = "h",
    showlegend=True,
    plot_bgcolor='white'
)
fig.show()
```

```
fig = px.line(trans_comp, x="dayofyear", y="gas_price", color = "year", hover_name= "dow",
fig.update_layout(
    xaxis=dict(
        title_text = "Day Of Year",
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        title_text = "Average Gas Price",
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=True,
    ),
```

```
        margin=dict(
            autoexpand=False,
            l=100,
            r=100,
            t=110,
        ),
        barmode = 'group',
        legend_orientation = "h",
        showlegend=True,
        plot_bgcolor='white'
    )
    fig.show()
```



```
    fig = px.line(trans_comp, x="dayofyear", y="receipt_status", color = "year", hover_name= "d
    fig.update_layout(
        xaxis=dict(
            title_text = "Day Of Year",
            showline=True,
```

```
            showgrid=False,
            showticklabels=True,
            linecolor='rgb(204, 204, 204)',
            linewidth=2,
            ticks='outside',
            tickfont=dict(
                family='Arial',
                size=12,
                color='rgb(82, 82, 82)',
            ),
        ),
        yaxis=dict(
            title_text = "Failure Count",
            showgrid=False,
            zeroline=False,
            showline=False,
            showticklabels=True,
        ),
        margin=dict(
            autoexpand=False,
            l=100,
            r=100,
            t=110,
        ),
        barmode = 'group',
        legend_orientation = "h",
        showlegend=True,
        plot_bgcolor='white'
    )
    fig.show()
```

2. Miners:

6 cells hidden

Contracts Table

12 cells hidden

Transaction Table

18 cells hidden