1. Problem Description

The company I work for heavily relies on fish resources as its major raw material. The fish are transported from all over the world to the United States. There are various types of costs associated with transportation and storage, such as demurrage, drayage, cold storage, material handling labor costs, and transportation costs to the factory. All of these costs are classified as variable overhead costs under the group named "fish handling cost," which accounts for approximately 10% of the overall conversion cost for the product. Therefore, an accurate fish handling cash flow forecast is crucial for the business operation and finance.

Given the importance of accurate forecasting, this is a perfect business use case for supervised machine learning technology. Below are the steps to implement this ML solution for fish handling cash forecasting.

2. . Data Discovery and Preparation:

Data Engineering and Architecture.

The volume of fish procured by our company is driven by production volume and schedule, and all related transactions are executed and maintained in our ERP system through the purchase order modules. Once the fish leaves its supplier's port, ownership is transferred to our company, and the transaction is marked as a "good issue" in the ERP system. Upon receipt of the fish at one of our cold storage locations, the transaction is marked as a "good receipt." All of these transactions are stored in the Material Movements tables in the ERP system's database, which drives the fish handling cost.

To determine the amount of cold storage fees charged each month, we must consider the inventory level. Therefore, historical beginning and ending inventory balance data must be collected. Additionally, knowing the production volume from previous periods can help with the ML modeling, as mentioned above.

Finally, the fish handling invoices data can provide insight into the cash being paid each month. With all of this data, we can create a reliable and accurate fish handling cash flow forecast.

Data Pipelines in Datawarehouse

1. Inventory Balance Snapshot: Every day, our ERP system's inventory management module captures a snapshot of the inventory balance. To bring this valuable data into our data warehouse, we have created a data pipeline that extracts the daily inventory snapshot and loads it into the warehouse.

2. Fish Inbound/Outbound Movements: The procurement of fish in our organization is facilitated through purchase orders. All inbound and outbound fish movements are recorded in the ERP system and stored in related tables. These tables have been identified in the ERP database and merged into a few tables in our data warehouse through a data pipeline.

3. Fish Invoice and Payments: All invoices received against fish receipts in our ERP system are recorded, and the payment transactions are associated with each invoice. These financial records have been extracted from the ERP system and loaded into our data warehouse using a data pipeline.

4. Production Volume: Every day, our ERP system stores production records at the production order level. To analyze this data in our data warehouse, we have developed a data pipeline that extracts all production records, aggregates them into monthly buckets, and loads them into a single table in the data warehouse. This makes it easier to analyze production volume trends over time.

EDA and Data Mining

1. Inventory Balance Data: Transformation is done in SQL

```sql
Select
i.[Material]
-- ,[Material Description]
-- ,try_cast([Plant] as nvarchar) as Plant
-- ,[Storage Location]
-- ,[Batch Number]
-- ,[Material Type]
,SnapshotDate
-- ,try_cast([Stock InQuality SU] as DECIMAL)
-- ,try_cast([InTransit Stock SU] as DECIMAL)
-- ,try_cast([Blocked Stock SU] as DECIMAL)
,Sum(try_cast([Total Stock SU] as DECIMAL)) as TotalQty
-- ,try_cast([RestrictedStock SU] as DECIMAL)
-- ,try_cast([Unrestricted Stock SU] as DECIMAL)
-- ,try_cast([Available] as DECIMAL)
-- ,try_cast([MAC Price] as DECIMAL)
,[UOM] as UoM
-- ,m.MATL_GROUP
-- ,mg.MaterialGRoupName as Mat_Grp_Desc
,d.FiscalPeriod
,d.FiscalPeriodWeek
from [SFS].[InventorySnapshots] i
Left JOIN [SC].[MaterialMaster] m
ON i.Material = m.MATERIAL
Left JOIN [GEN].[DimMaterialGroup] mg
ON m.MATL_GROUP = mg.MaterialGroupID
Left Join [GEN].[DimDate] d
ON d.[FullDate]=i.SnapshotDate
WHere m.MATL_GROUP in ('NF17','NF28') and Year([SnapshotDate])>=Year(Getdate()-1)-3 and[Storage Location]='WHSE'
Group By d.FiscalPeriod,i.Material
,d.FiscalPeriodWeek,SnapshotDate,UOM
```

Features are the fish material#, Fish material Description, Material group , snapshot date and fiscal Period.

Based on the snapshot date, the Inventory with min of snapshot date in the same fiscal period is labeled beginning inventory for the fiscal period. the Inventory with max of snapshot date in the same fiscal period is labeled ending inventory for the fiscal period.

Load fish inventory data into Panda dataframe, total the quantity by snapshot date then convert the unit of measure to metric ton.

```python
df = pd.read_excel(r'C:\Users\ilin\Downloads\OFC_1_FishInventory.xlsx')
df=df.groupby(['FiscalPeriod','FiscalPeriodWeek','SnapshotDate'],as_index=False)['TotalQty'].sum()
# df=df.to_frame() #turn searise data into DF

df=df.groupby(['FiscalPeriod','FiscalPeriodWeek'],as_index=False)['TotalQty'].mean()
df['TtlMT']=df['TotalQty']/2204.6
df.tail()
# print(df['FiscalPeriod'])
```

5]:

	FiscalPeriod	FiscalPeriodWeek	TotalQty	TtlMT
155	2023001	3	10099255.0	4580.992017
156	2023001	4	10170984.0	4613.528078
157	2023002	1	10146631.0	4602.481629
158	2023002	2	10713701.0	4859.702894
159	2023002	3	10835095.0	4914.766851

Transform the inventory data to weeks pivottable layout. this makes the beginning and ending inventory logic easy to implement.

```
df= df.pivot(index=['FiscalPeriod'],columns='FiscalPeriodWeek',values='TtlMT')
# df= df.mean()
```

```
df.head()
```

[7]:

FiscalPeriodWeek	1	2	3	4	5
FiscalPeriod					
2020001	3049.027488	3536.496417	3103.600200	3121.676495	NaN
2020002	3119.687925	2695.267169	2359.744625	2057.421301	NaN
2020003	2316.350812	2441.844779	2053.261816	2105.208201	2035.600109
2020004	2542.743355	NaN	2694.276966	2708.263631	NaN
2020005	2941.140797	2150.299828	2207.898485	2367.375941	NaN

Beginning and Ending Inventory Calculation. Column 1 is beginning inventory, and column 4 is the ending, if column 5 is empty.

```
df['BegInv']=df[1]
df['EndInv'] = df.apply(lambda x: x[5] if x[5] > 0 else x[4], axis=1)
df_Inv=df
df_Inv.head()
```

8]:

FiscalPeriodWeek	1	2	3	4	5	BegInv	EndInv
FiscalPeriod							
2020001	3049.027488	3536.496417	3103.600200	3121.676495	NaN	3049.027488	3121.676495
2020002	3119.687925	2695.267169	2359.744625	2057.421301	NaN	3119.687925	2057.421301
2020003	2316.350812	2441.844779	2053.261816	2105.208201	2035.600109	2316.350812	2035.600109
2020004	2542.743355	NaN	2694.276966	2708.263631	NaN	2542.743355	2708.263631
2020005	2941.140797	2150.299828	2207.898485	2367.375941	NaN	2941.140797	2367.375941

2. Fish Inbound/Outbound Movement data:

```sql
SELECT a.[Material]
      ,[Posting Date]
  --    ,[Plant]
      ,[Fiscal Period]
      ,[Fiscal Year]
  --    ,[Fiscal Variant]
  --    ,[Material Document Number]
  --    ,[Material Document Item]
  --    ,[Customer]
      ,[Batch]
  --    ,[Vendor]
      ,[Movement Type]
      ,[Storage Location]
      ,[PO Number]
      ,[Quantity]
      ,[Unit of Measure]
  --    ,[Sales Order Number]
  --    ,[Sales Order Item]
  --    ,[Currency]
  --    ,[Amount]
  --    ,[GL Account]
  --    ,[Chart of Accounts]
  --    ,[Company Code]
  --    ,[Reference Document Number]
  --    ,[Cost Center]
  --    ,[Controlling Area]
  --    ,[Long Description]
  --    ,[Order Number]
      ,[Name]
  --    ,b.EQ
  FROM [SC].[MaterialMovements] a
  INNER Join (Select MATERIAL From [SC].[MaterialMaster]
              Where [MATL_GROUP] in ('NF17','NF28')) b
  ON a.[Material]= b.MATERIAL--(REPLICATE('0',18-len(b.MaterialId))+b.MaterialId)
  Where Plant=1100 and [Fiscal Year]>=Year(Getdate())-3
  and [Movement Type] in ('351','352') and [PO Number] is not null
```

Features are [Material] ,[Posting Date],[Fiscal Period],[Fiscal Year] ,[Batch] ,[Movement Type],[Storage Location] ,[PO Number] ,[Quantity] ,[Unit of Measure]

Data was pulled with filters of specific material movement types, 351 and 352; and specific material groups, NF17 and NF28.

Load fish movements data that is downloaded from the data warehouse.

```python
df_Fishmove = pd.read_excel(r'C:\Users\ilin\Downloads\OFC_5_LoinMoves.xlsx')
df_Fishmove=df_Fishmove[['Fiscal Period','Quantity','Unit of Measure']]
df_Fishmove
```

]:

	Fiscal Period	Quantity	Unit of Measure
0	2021007	-2731.499	LB
1	2021007	-2821.888	LB
2	2021007	-2696.226	LB
3	2020001	-1.000	LB
4	2020005	-2563.950	LB
...	...	...	...
108040	2022012	2623.474	LB
108041	2022012	2623.474	LB
108042	2022012	2623.474	LB
108043	2022012	2623.474	LB
108044	2022012	2623.474	LB

108045 rows × 3 columns

Fish movement is aggerated to fiscal period level in metric tons.

```
df_Fishmove['Quantity']= df_Fishmove.apply(lambda x: x['Quantity']/2204.6 if x['Unit of Measure'] =='LB' else x['Quantity'],
df_Fishmove=df_Fishmove.groupby(['Fiscal Period'],as_index=False)['Quantity'].sum()
df_Fishmove.head()
```

]:

	Fiscal Period	Quantity
0	2020001	1157.993992
1	2020002	1868.092002
2	2020003	1617.028000
3	2020004	1440.304001
4	2020005	1340.953000

3. Fish Invoice and Payments data:

```sql
SELECT [InvoiceNumber]
      ,[InvoiceItem]
      ,[Material]
      ,[Plant]
      ,[Vendor]
      ,[MaterialType]
      ,[PostingDate]
      ,[FiscalYearPeriod]
      ,[Amount]
  FROM [SFS].[PurchaseOrderInvoices]
  Where [MaterialGroup] in ('NF17','NF28')
  and ClearingDocument='' and GoodsReceiptDate<>'00000000'
```

Features are [InvoiceNumber], [InvoiceItem], [Material] ,[Plant], [Vendor] , [MaterialType] ,[PostingDate],[FiscalYearPeriod],[Amount] with filters of and specific material groups, NF17 and NF28

Load Invoice and payment data that was downloaded from the data warehouse. Three years' data was saved into three excel files.

```python
files = [r'C:\Users\ilin\Downloads\OFC_2_2022Payment.xlsx',
         r'C:\Users\ilin\Downloads\OFC_3_2021Payment.xlsx',
         r'C:\Users\ilin\Downloads\OFC_4_2023Payment.xlsx']

df_Pmt = pd.concat([pd.read_excel(f) for f in files])
df_Pmt = df_Pmt.dropna(subset=['Text'])
df_Pmt = df_Pmt[df_Pmt['Text'].str.contains("Accrual") & (df_Pmt['Amount in local cur.'] > 0)]
df_Pmt.head()
```

[9]:

	DocumentNo	Account	CoCd	Profit Ctr	Vendor Name	User name	Text	Reference	Pstng Date	Amount in local cur.
122	102456920	540474	1001	11000	NaN	SGOMEZ	OFC Accrual P1	840111	2022-01-29	164004.46
231	102461654	540474	1001	11000	NaN	SGOMEZ	OFC Accrual P2	840210	2022-02-25	219597.91
295	102484392	540474	1001	11000	NaN	SGOMEZ	OFC Accrual P3	840314	2022-04-01	283188.20
413	102486705	540474	1001	11000	NaN	SGOMEZ	OFC Accrual P4	840406	2022-04-29	466399.35
513	102492506	540474	1001	11000	NaN	SGOMEZ	OFC Accrual P5	840507	2022-05-27	595096.53

```python
Cal = pd.read_excel(r'C:\Users\ilin\Downloads\MasterCalendar.xlsx')
```

```python
df_Pmt = df_Pmt.merge(Cal, left_on='Pstng Date', right_on='Date', how='left')
df_Pmt.head()
```

t	CoCd	Profit Ctr	Vendor Name	User name	Text	Reference	Pstng Date	Amount in local cur.	Date	WeekNumber	Quarter	Period	PeriodWeek	FiscPd
	1001	11000	NaN	SGOMEZ	OFC Accrual P1	840111	2022-01-29	164004.46	2022-01-29	4	1	1	4	2022001
	1001	11000	NaN	SGOMEZ	OFC Accrual P2	840210	2022-02-25	219597.91	2022-02-25	8	1	2	4	2022002
	1001	11000	NaN	SGOMEZ	OFC Accrual P3	840314	2022-04-01	283188.20	2022-04-01	13	1	3	5	2022003
	1001	11000	NaN	SGOMEZ	OFC Accrual P4	840406	2022-04-29	466399.35	2022-04-29	17	2	4	4	2022004
	1001	11000	NaN	SGOMEZ	OFC Accrual P5	840507	2022-05-27	595096.53	2022-05-27	21	2	5	4	2022005

```python
df_Pmt=df_Pmt[['FiscPd','Amount in local cur.']]
df_Pmt.head()
```

	FiscPd	Amount in local cur.
0	2022001	164004.46
1	2022002	219597.91
2	2022003	283188.20
3	2022004	466399.35
4	2022005	595096.53

4. Production Volume:
   Data is directly pulled from an existing Power BI dashboard which is using SQL pool from our data warehouse.
   It is leveraging the existing data transformation procedures, to avoid duplicate efforts.
   Features are Fiscal Period, and Actual Qty

Load production volume data into data frame

```python
df_Prd = pd.read_excel(r'C:\Users\ilin\Downloads\OFC_6_Production.xlsx')
df_Prd.head()
```

5]:

	Row Labels	ActStdCases
0	2020001	656337
1	2020002	641157
2	2020003	861286
3	2020004	739870
4	2020005	720804

3. Machine Learning Data Modeling

After data processing steps and EDA, all the datasets are merged into one flat table.

Final dataset- merge all the datasets from above into one flat table

```
df_Inv = df.reset_index()
df = df_Inv[['FiscalPeriod','BegInv','EndInv']].set_index('FiscalPeriod')

df=df.merge(df_Pmt, left_on='FiscalPeriod', right_on='FiscPd', how='inner')
df=df.merge(df_Fishmove, left_on='FiscPd', right_on='Fiscal Period', how='left')
df=df.merge(df_Prd, left_on='FiscPd', right_on='Row Labels', how='left')
df=df[['Fiscal Period','BegInv','EndInv','Amount in local cur.','Quantity','ActStdCases']]
df.columns=['Fiscal Period','BegInv','EndInv','Payment','MoveQty','StdCases']
df
```

16]:

	Fiscal Period	BegInv	EndInv	Payment	MoveQty	StdCases
0	2020001	3049.027488	3121.676495	248997.6875	1157.993992	656337
1	2020002	3119.687925	2057.421301	195793.5625	1868.092002	641157
2	2020003	2316.350812	2035.600109	196274.8750	1617.028000	861286

Analysis steps:

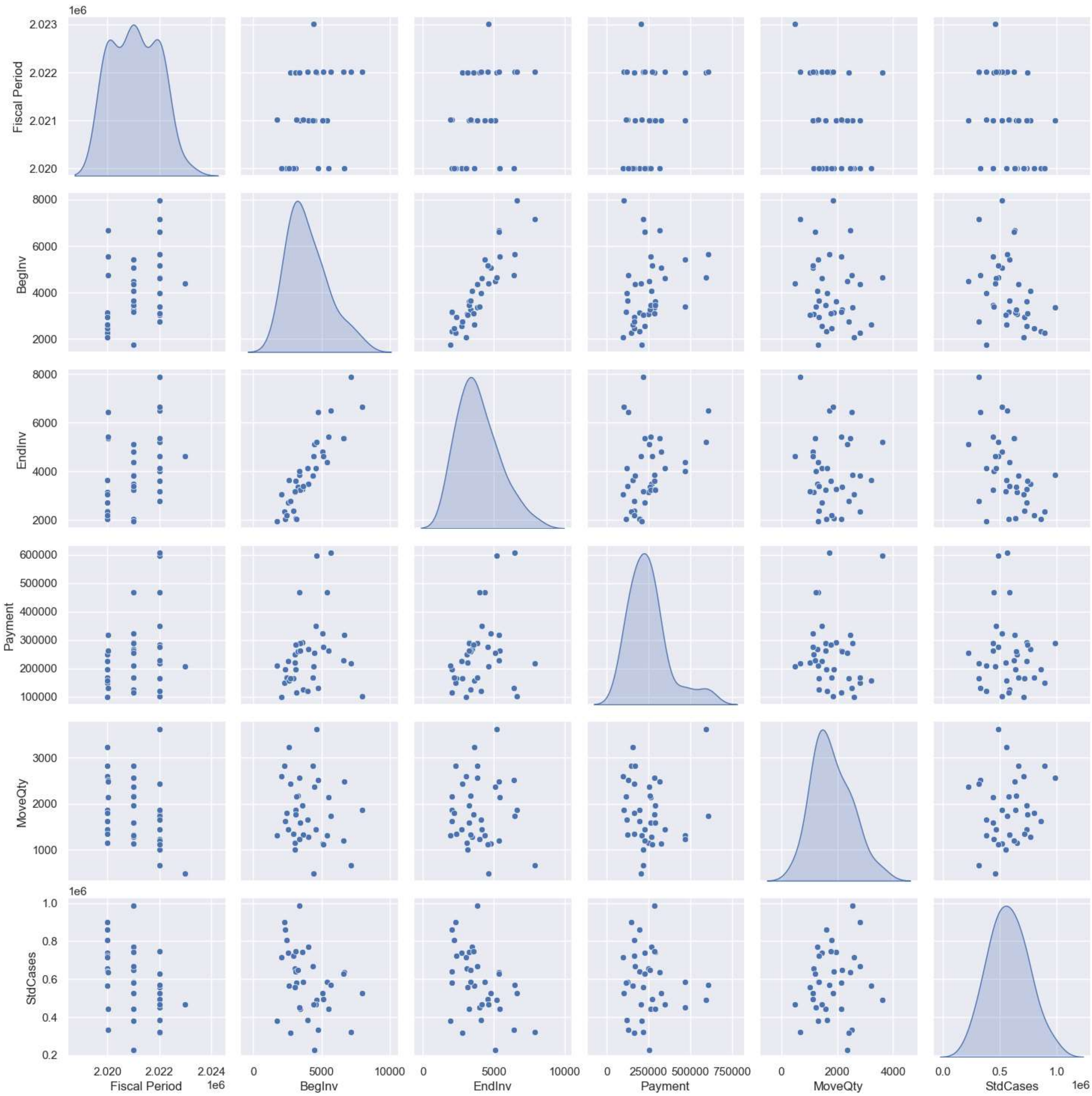
1. Multilinear Regression: since this dataset has multiple independent variables (Beginning Inventory, Ending Inventory, Production cases, Movement Qty) and a single dependent variable (payment)
2. If the accuracy is low and the data shows non-linear relationships with all the variables , then will do polynomial regression.
3. If Polynomial is not ideal, will do random forest regression. Random forest regression is robust to overfitting, which occurs when a model is too complex and fits the noise in the data instead of the underlying signal. This is because the algorithm builds multiple trees, each of which is trained on a random subset of the data, which helps to reduce overfitting.
4. Lastly, will do an AdaBoost regression. The AdaBoost regression algorithm works by building a sequence of weak regression models, each of which is trained on a subset of the data and assigned a weight based on its performance. The algorithm then combines the predictions of all the models to make a final prediction. The weights of the models are adjusted based on the error of the previous model, so that the subsequent models focus more on the data points that were incorrectly predicted by the previous models.

In the end, the final method will be based on the accuracy score from each technics from the above.

First, started with the testing the correlation between each variables. The correlation matrix is plotted in the chart below.

```
df.corr()
sns.pairplot(df, diag_kind = 'kde')
plt.savefig('pair_plot.png', dpi = 300, bbox_inches = 'tight')
```

Corrlation Matrix



Run multilinear regression model

```
model= smf.ols(formula= 'Payment~BegInv + EndInv + MoveQty+StdCases ', data=df).fit()
model.summary()
```

18]:

OLS Regression Results

Dep. Variable:	Payment	**R-squared:**	0.109
Model:	OLS	**Adj. R-squared:**	0.001
Method:	Least Squares	**F-statistic:**	1.012
Date:	Sat, 25 Feb 2023	**Prob (F-statistic):**	0.415
Time:	00:15:57	**Log-Likelihood:**	-495.77
No. Observations:	38	**AIC:**	1002.
Df Residuals:	33	**BIC:**	1010.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	9.862e+04	1.25e+05	0.791	0.435	-1.55e+05	3.52e+05
BegInv	-8.9613	30.164	-0.297	0.768	-70.330	52.407
EndInv	38.3806	32.533	1.180	0.247	-27.808	104.570
MoveQty	-2.0636	29.913	-0.069	0.945	-62.923	58.795
StdCases	0.0679	0.133	0.510	0.613	-0.203	0.339

Omnibus:	7.609	**Durbin-Watson:**	1.122
Prob(Omnibus):	0.022	**Jarque-Bera (JB):**	6.266
Skew:	0.846	**Prob(JB):**	0.0436
Kurtosis:	4.046	**Cond. No.**	3.85e+06

Based on the correlation matrix and the results from the multi-linear model, it shows the variables are not in a linear relationship. The r-squared score is only 0.109. that suggests we should run polynomial regression.

Run Polynomial regression:

Build an iteration to run through 20 degrees.

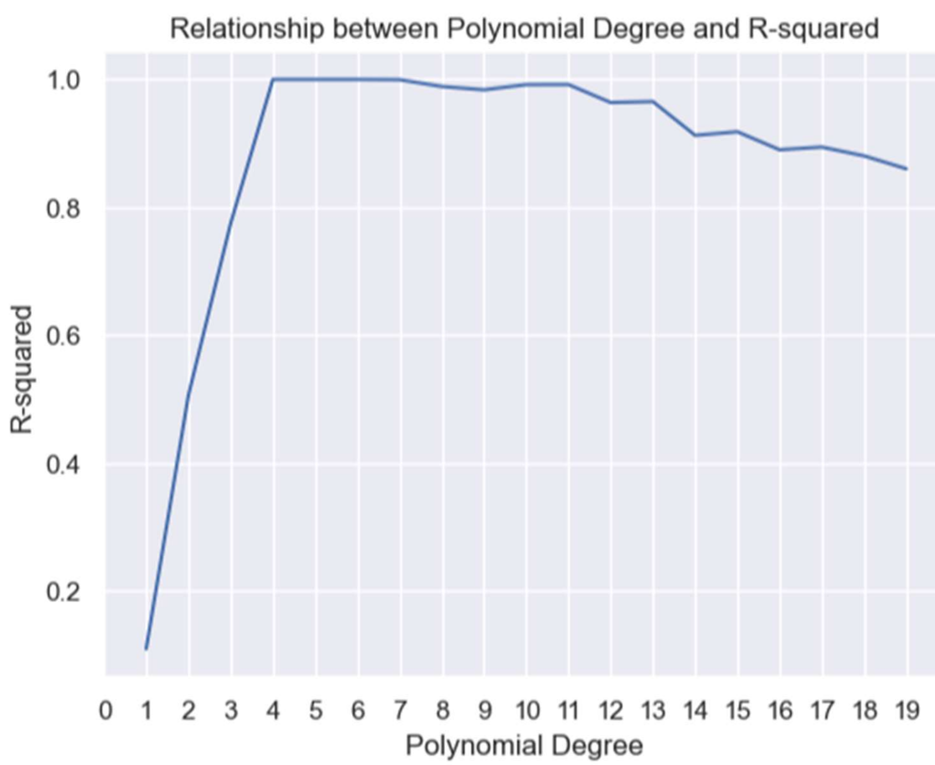
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics

X=df[['BegInv','EndInv','MoveQty','StdCases']]
y=df['Payment']
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
r2=[]
l=[]
for i in range(1,20):
    poly = PolynomialFeatures(degree=i)
    X_poly = poly.fit_transform(X)

    # Fit linear regression model
    reg = LinearRegression().fit(X_poly, y)
    y_pred = reg.predict(X_poly)
    r2.append(metrics.r2_score(y, y_pred))
    l.append(i)
# print("Accuracy of the model:", metrics.r2_score(y, y_pred))
```

```
print(max(r2))
fig, ax = plt.subplots()
ax.plot(l,r2)
# plt.plot(l,r2)
plt.xlabel('Polynomial Degree')
plt.ylabel('R-squared')
plt.title('Relationship between Polynomial Degree and R-squared')
ax.set_xticks(np.arange(0, 20, 1))
plt.show()
```

```
0.9994176077541682
```



The best R-squared score is from the 4th degree, based on the plot chart, it is apparently overfitted. In this case, the 3rd degree is chosen for the model. It has an accuracy score of 0.774

```
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X)

# Fit linear regression model
reg_yes = LinearRegression().fit(X_poly, y)
y_pred = reg_yes.predict(X_poly)
print("Accuracy of the model:", metrics.r2_score(y, y_pred))
```

```
Accuracy of the model: 0.7743487239098648
```

In order to further improve the accuracy, a random forest model is constructed.

```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
```

```
rf_train=rf.fit(X, y)
rf_pred=rf.predict(X)
print("Accuracy of the model:", metrics.r2_score(y, rf_pred))
```

```
Accuracy of the model: 0.8261013602731134
```

The accuracy score is improved to 0.826

Lastly constructed an Adaboost regression model

```
from sklearn.ensemble import AdaBoostRegressor
ada = AdaBoostRegressor()

# fit the model on the training data
ada.fit(X,y)
ada_pred=ada.predict(X)
print("Accuracy of the model:", metrics.r2_score(y, ada_pred))
```

```
Accuracy of the model: 0.8731009548456359
```

The accuracy score is improved to 0.873, which exceeds the target of 0.850.

```
# New data for which predictions are to be made
new_data = pd.DataFrame({'BegInv': [4250], 'EndInv': [4220],
                         'MoveQty': [1862], 'StdCases': [526576]})

# new_data = pd.DataFrame({'BegInv': [12350], 'EndInv': [11940],
#                          'MoveQty': [1312], 'StdCases': [584822]})

# Make predictions on new data
predictions = rf.predict(new_data)

print("Predictions:", predictions)
```

```
Predictions: [250284.57195833]
```

Conclusion :

Based on the analysis of the dataset, four different regression models were used to predict the cashflow variable-payment. The accuracy score for the four models are.

1. Multi-linear regression : 0.109
2. Polynomial regression: 0.774
3. Random forest: 0.826
4. AdaBoost: 0.873

After evaluating the accuracy scores of all four models, it was found that AdaBoost Regression had the highest accuracy score. Therefore, it can be concluded that AdaBoost Regression is the most appropriate model for predicting the fish handling cost in this dataset.