In [34]:
```
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import seaborn as sns
from factor_analyzer import FactorAnalyzer
from factor_analyzer.factor_analyzer import calculate_kmo
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import KNNImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import xgboost as xgb
from sklearn.model_selection import train_test_split
```

Preparación de los datos

In [35]:
```
data = pd.read_csv("customer_satisfaction_data.csv")
data
```

Out[35]:

	customer_id	quarter	survey_date	account_manager_responsive	billing_accuracy	bu
0	CUST_001	Q1_2024	2024-03-22	5.0	6.0	
1	CUST_002	Q1_2024	2024-03-20	5.0	3.0	
2	CUST_003	Q1_2024	2024-03-17	4.0	3.0	
3	CUST_004	Q1_2024	2024-03-08	3.0	4.0	
4	CUST_005	Q1_2024	2024-03-12	5.0	5.0	
...	...	...	...	...	...	
3395	CUST_846	Q4_2024	2024-12-16	4.0	6.0	
3396	CUST_847	Q4_2024	2024-12-14	3.0	5.0	
3397	CUST_848	Q4_2024	2024-12-16	3.0	3.0	
3398	CUST_849	Q4_2024	2024-12-19	5.0	4.0	
3399	CUST_850	Q4_2024	2024-12-10	3.0	4.0	

3400 rows × 31 columns

In [36]:
```
print(f"Variables: {data.shape[1]}, Observaciones: {data.shape[0]}")
print("Tipos de datos:")
print(data.dtypes.value_counts())
```

```
Variables: 31, Observaciones: 3400
Tipos de datos:
float64    24
int64       4
object      3
Name: count, dtype: int64
```

In [37]:
```python
# Definir las variables por dimension
technical_vars = ['technical_expertise', 'problem_solving', 'innovation_solutions',
                  'technical_documentation', 'system_integration']

relationship_vars = ['account_manager_responsive', 'executive_access', 'trust_relia
                     'long_term_partnership', 'communication_clarity']

project_vars = ['project_management', 'timeline_adherence', 'budget_control',
                'quality_deliverables', 'change_management']

value_vars = ['cost_transparency', 'value_for_money', 'roi_demonstration',
              'competitive_pricing', 'billing_accuracy']

support_vars = ['support_responsiveness', 'training_quality', 'documentation_help']

outcome_vars = ['overall_satisfaction', 'nps_score', 'renewal_likelihood',
                'revenue_growth_pct', 'referrals_generated']

# Combinar todas las variables de satisfacción
satisfaction_vars = technical_vars + relationship_vars + project_vars + value_vars
numerical_vars = satisfaction_vars + outcome_vars

print(f"\nTotal variables de satisfacción: {len(satisfaction_vars)}")
print(f"Total variables de outcome: {len(outcome_vars)}")
```

```
Total variables de satisfacción: 23
Total variables de outcome: 5
```

In [38]:
```python
# VALORES NULOS
print(data.isnull().sum())
```

```
customer_id                   0
quarter                       0
survey_date                   0
account_manager_responsive    16
billing_accuracy              13
budget_control                17
change_management             12
communication_clarity         16
competitive_pricing           15
cost_transparency             10
documentation_help            14
executive_access              10
innovation_solutions          10
long_term_partnership         14
problem_solving               20
project_management            17
quality_deliverables          13
roi_demonstration             17
support_responsiveness        17
system_integration            14
technical_documentation       19
technical_expertise            9
timeline_adherence             8
training_quality              17
trust_reliability             22
value_for_money               15
overall_satisfaction           0
nps_score                      0
renewal_likelihood             0
revenue_growth_pct             0
referrals_generated            0
dtype: int64
```

In [39]:
```python
#Inputacion de datos nulos usando KNNImputer (Metodo de imputacion propuesto por ge
data_with_index = data.set_index('customer_id')

data_to_impute = data_with_index[numerical_vars]

original_columns = data_to_impute.columns
original_index = data_to_impute.index

imputer = KNNImputer(n_neighbors=5)
data_imputed_array = imputer.fit_transform(data_to_impute)

# Reconstruye el DataFrame imputado (con las 23+5 columnas)
data_imputed = pd.DataFrame(
    data_imputed_array,
    columns=original_columns,
    index=original_index
)

#Nuevos valores nulos
print(data_imputed.isnull().sum())
```

```
technical_expertise           0
problem_solving               0
innovation_solutions          0
technical_documentation       0
system_integration            0
account_manager_responsive    0
executive_access              0
trust_reliability             0
long_term_partnership         0
communication_clarity         0
project_management            0
timeline_adherence            0
budget_control                0
quality_deliverables          0
change_management             0
cost_transparency             0
value_for_money               0
roi_demonstration             0
competitive_pricing           0
billing_accuracy              0
support_responsiveness        0
training_quality              0
documentation_help            0
overall_satisfaction          0
nps_score                     0
renewal_likelihood            0
revenue_growth_pct            0
referrals_generated           0
dtype: int64
```

Exploración de los datos

In [40]:
```
# ESTADISTICAS DESCRIPTIVAS
print(data.describe())
```

	account_manager_responsive	billing_accuracy	budget_control
count	3384.000000	3387.000000	3383.000000
mean	4.115248	4.101270	4.100798
std	0.970303	0.962109	0.983397
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	3.000000
50%	4.000000	4.000000	4.000000
75%	5.000000	5.000000	5.000000
max	7.000000	7.000000	7.000000

	change_management	communication_clarity	competitive_pricing
count	3388.000000	3384.000000	3385.000000
mean	4.109504	4.092494	4.079468
std	0.977813	0.964956	0.979341
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	3.000000
50%	4.000000	4.000000	4.000000
75%	5.000000	5.000000	5.000000
max	7.000000	7.000000	7.000000

	cost_transparency	documentation_help	executive_access
count	3390.000000	3386.000000	3390.000000
mean	4.100885	4.079445	4.113569
std	0.981008	0.979800	0.969478
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	4.000000
50%	4.000000	4.000000	4.000000
75%	5.000000	5.000000	5.000000
max	7.000000	7.000000	7.000000

	innovation_solutions	...	technical_expertise	timeline_adherence
count	3390.000000	...	3391.000000	3392.000000
mean	4.105310	...	4.115305	4.110554
std	0.985496	...	0.974591	0.981026
min	1.000000	...	1.000000	1.000000
25%	3.000000	...	4.000000	3.000000
50%	4.000000	...	4.000000	4.000000
75%	5.000000	...	5.000000	5.000000
max	7.000000	...	7.000000	7.000000

	training_quality	trust_reliability	value_for_money
count	3383.000000	3378.000000	3385.000000
mean	4.072421	4.100355	4.091581
std	0.998707	0.961346	0.975105
min	1.000000	1.000000	1.000000
25%	3.000000	3.000000	3.000000
50%	4.000000	4.000000	4.000000
75%	5.000000	5.000000	5.000000
max	7.000000	7.000000	7.000000

	overall_satisfaction	nps_score	renewal_likelihood
count	3400.000000	3400.000000	3400.000000
mean	4.125588	6.169118	3.070294
std	0.817824	1.777549	0.761728
min	1.000000	0.000000	1.000000
25%	4.000000	5.000000	3.000000

```
50%                4.000000     6.000000           3.000000
75%                5.000000     7.000000           4.000000
max                7.000000    10.000000           5.000000

       revenue_growth_pct  referrals_generated
count         3400.000000          3400.000000
mean             6.072176             1.615588
std              8.315453             1.510634
min            -24.700000             0.000000
25%              0.300000             0.000000
50%              6.150000             1.000000
75%             11.700000             2.000000
max             40.100000             9.000000

[8 rows x 28 columns]
```

In [41]:
```python
# Visualizacion de variables usando Plotly
def create_interactive_boxplots(data, variable_groups, titles):
    n_rows = 2
    n_cols = 3
    fig = make_subplots(
        rows=n_rows, cols=n_cols,
        subplot_titles=titles,
        vertical_spacing=0.12
    )

    iter = 0
    for r in range(n_rows):
        for c in range(n_cols):
            # Grupo X
            for i, var in enumerate(variable_groups[iter]):
                fig.add_trace(
                    go.Box(y=data[var], name=var, showlegend=False),
                    row=r+1, col=c+1
                )
            iter+=1

    fig.update_layout(
        height=1200,
        title_text="Distribución de Variables - Análisis de Satisfacción",
        showlegend=False
    )

    return fig

# visualizacion
variable_groups = [
    technical_vars,
    relationship_vars,
    project_vars,
    value_vars,
    support_vars,
    outcome_vars
]

titles = [
```

```
    'Distribución - Excelencia Técnica',
    'Distribución - Gestión de Relaciones',
    'Distribución - Entrega de Proyectos',
    'Distribución - Costo y valor',
    'Distribución - Soporte y Servicio',
    'Distribución - Variables de Outcome',
]

fig = create_interactive_boxplots(data_imputed, variable_groups, titles)
fig.show()
```

Validación de ideoneidad de los datos

In [42]:
```
# Matriz de correlación para variables de satisfacción
correlation_matrix = data_imputed[satisfaction_vars].corr()

# heatmap con Plotly
fig = px.imshow(
    correlation_matrix,
    x=correlation_matrix.columns,
    y=correlation_matrix.index,
    color_continuous_scale='RdBu_r',
    zmin=-1, zmax=1,
    aspect="auto"
)

fig.update_layout(
    title='Matriz de Correlación Variables de Satisfaccion',
    width=800,
    height=800,
    xaxis_title="Variables",
    yaxis_title="Variables"
)

fig.show()
```

In [43]:
```
# Test de KMO (ayuda de Deeepseek)
def check_factor_analysis_suitability(data, variables):
    """Verificar la adecuación del análisis factorial"""

    # Limpiar y calcular KMO
    data_clean = data[variables].fillna(data[variables].median())
    kmo_all, kmo_model = calculate_kmo(data_clean)

    print(f"KMO Overall: {kmo_model:.3f}")

    # Visualización
    kmo_df = pd.DataFrame({'Variable': variables, 'KMO': kmo_all}).sort_values('KMO

    fig = px.bar(kmo_df, x='KMO', y='Variable', orientation='h',
                 title='KMO por Variable', color='KMO')
    fig.add_vline(x=0.6, line_dash="dash", line_color="red")
    fig.show()
```

```
    # Evaluación
    adequacy = "Adecuado" if kmo_model > 0.6 else "Inadecuado"
    print(f"Evaluación: {adequacy}")

    return kmo_model, kmo_df

# Aplicar test
kmo_overall, kmo_variables = check_factor_analysis_suitability(data_imputed, satisf
```

```
KMO Overall: 0.959
Evaluación: Adecuado
```

In [44]:

```
# evaluacion de correlaciones
def assess_correlations(data, variables, threshold=0.3):
    corr_matrix = data[variables].corr()

    # Porcentaje de correlaciones significativas
    significant_corrs = (abs(corr_matrix) >= threshold).sum().sum() - len(variables
    total_possible_corrs = len(variables) * (len(variables) - 1)
    percent_significant = (significant_corrs / total_possible_corrs) * 100

    print(f"Umbral de correlación: |r| ≥ {threshold}")
    print(f"Correlaciones significativas: {significant_corrs}/{total_possible_corrs
    print(f"Porcentaje: {percent_significant:.1f}%")

    # Visualización de distribución de correlaciones
    corr_values = corr_matrix.values[np.triu_indices_from(corr_matrix.values, k=1)]

    fig = px.histogram(
        x=corr_values,
        nbins=50,
        title='Distribución de Correlaciones entre Variables',
        labels={'x': 'Coeficiente de Correlación', 'y': 'Frecuencia'}
    )

    fig.add_vline(x=threshold, line_dash="dash", line_color="red",
                  annotation_text=f"Umbral {threshold}")
    fig.add_vline(x=-threshold, line_dash="dash", line_color="red")

    fig.update_layout(showlegend=False)
    fig.show()

    # Matriz de correlaciones
    corr_det = np.linalg.det(corr_matrix)
    print(f"Determinante de la matriz de correlación: {corr_det:.6f}")

    # Test de esfericidad de Bartlett (aproximado)
    n = len(data)
    p = len(variables)
    chi_square = -((n - 1) - (2 * p + 5) / 6) * np.log(corr_det)
    df = p * (p - 1) / 2

    print(f"Chi-cuadrado aproximado: {chi_square:.2f}")
    print(f"Grados de libertad: {df}")
```

```
    return percent_significant

# Evaluar correlaciones
corr_percentage = assess_correlations(data_imputed, satisfaction_vars)
```

```
Umbral de correlación: |r| ≥ 0.3
Correlaciones significativas: 244/506
Porcentaje: 48.2%
Determinante de la matriz de correlación: 0.000037
Chi-cuadrado aproximado: 34625.63
Grados de libertad: 253.0
```

El análisis de adecuación para la aplicación del análisis factorial mostró resultados favorables. Tenemos un KMO de 0.959, superando el umbral mínimo recomendado de 0.6, lo que indica una alta adecuación muestral y sugiere que las variables comparten una cantidad suficiente de varianza común para justificar el uso de este tipo de análisis. Tambien el 48.2% de las correlaciones entre las variables presentaron valores iguales o superiores a 0.3, también supera el mínimo sugerido del 30%, mostrandonos relaciones significativas entre los ítems. Ambos indicadores confirman que los datos son adecuados para proceder con un análisis factorial.

Extracción y determinación de factores

In [45]:
```
# Estandarizar los datos para el análisis factorial
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_imputed[satisfaction_vars])
data_scaled = pd.DataFrame(data_scaled, columns=satisfaction_vars, index=original_i

# componentes principales
fa = FactorAnalyzer(n_factors=len(satisfaction_vars), rotation=None)
fa.fit(data_scaled)

# Obtener eigenvalues
ev, v = fa.get_eigenvalues()
ev
```

```
c:\Users\crdie\anaconda3\envs\cienciaDatos\Lib\site-packages\sklearn\utils\deprecati
on.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in
1.8.
```

Out[45]:
```
array([8.69671635, 1.78545446, 1.44302724, 1.20242047, 1.07347165,
       0.75944407, 0.74470045, 0.64905299, 0.64368785, 0.60632468,
       0.58393864, 0.49774711, 0.47032578, 0.46464653, 0.464051  ,
       0.40689684, 0.40134693, 0.3961113 , 0.37204779, 0.35713056,
       0.33971811, 0.3257686 , 0.31597059])
```

In [46]:
```
# grafico de scree plot (tambien un poco de ayuda de Deepseek para correccion de er
def create_interactive_scree_plot(eigenvalues, variance_proportion):
    cumulative_variance = np.cumsum(variance_proportion)
    factors = list(range(1, len(eigenvalues) + 1))
```

```
    fig = make_subplots(
        rows=1, cols=2,
        subplot_titles=('Scree Plot', 'Varianza Acumulada Explicada'),
        horizontal_spacing=0.1
    )

    # Scree plot
    fig.add_trace(
        go.Scatter(x=factors, y=eigenvalues, mode='lines+markers',
                   name='Eigenvalues', line=dict(width=3)),
        row=1, col=1
    )
    fig.add_trace(
        go.Scatter(x=factors, y=[1]*len(factors), mode='lines',
                   name='Umbral Eigenvalue = 1', line=dict(dash='dash', color='red')
        row=1, col=1
    )

    # Varianza acumulada
    fig.add_trace(
        go.Scatter(x=factors, y=cumulative_variance, mode='lines+markers',
                   name='Varianza Acumulada', line=dict(width=3, color='green')),
        row=1, col=2
    )
    fig.add_trace(go.Scatter(x=factors, y=[0.7]*len(factors), mode='lines',
                             name='70% Varianza', line=dict(dash='dash', color='red
                   row=1, col=2)
    fig.add_trace(go.Scatter(x=factors, y=[0.8]*len(factors), mode='lines',
                             name='80% Varianza', line=dict(dash='dash', color='ora
                   row=1, col=2)

    fig.update_xaxes(title_text="N_Factores", row=1, col=1)
    fig.update_xaxes(title_text="N_Factores", row=1, col=2)
    fig.update_yaxes(title_text="Eigenvalor", row=1, col=1)
    fig.update_yaxes(title_text="Varianza acumulada", row=1, col=2)

    fig.update_layout(height=500, title_text="Determinación del Número Óptimo de Fa
    return fig



variance_explained = fa.get_factor_variance()
variance_proportion = variance_explained[0] / len(satisfaction_vars)
cumulative_variance = np.cumsum(variance_proportion)

fig = create_interactive_scree_plot(ev, variance_proportion)
fig.show()
```

In [66]:

```
def determine_optimal_factors(eigenvalues, variance_proportion, cumulative_variance
    kaiser_factors = sum(eigenvalues > 1)
    print(f"Criterio de Kaiser (eigenvalues > 1): {kaiser_factors} factores")
    print(f"Eigenvalues: {[f'{ev:.3f}' for ev in eigenvalues[:10]]}...")

    fig = px.bar(
```

```
        x=list(range(1, len(eigenvalues)+1)),
        y=eigenvalues,
        title='Eigenvalues por Factor (Criterio de Kaiser)',
        labels={'x': 'Número de Factor', 'y': 'Eigenvalue'}
    )
    fig.add_hline(y=1, line_dash="dash", line_color="red", annotation_text="Umbral
    fig.show()

    print("Varianza explicada:")
    for i, (var, cum_var) in enumerate(zip(variance_proportion, cumulative_variance
        if i < 10:
            print(f"Factor {i+1}: {var:.3f} ({cum_var:.3f} acumulada)")

    factors_70 = np.argmax(cumulative_variance >= 0.7) + 1
    factors_80 = np.argmax(cumulative_variance >= 0.8) + 1

    print(f"Factores para 70% de varianza: {factors_70}")
    print(f"Factores para 80% de varianza: {factors_80}")

    differences = np.diff(eigenvalues)
    inflection_point = np.argmin(differences) + 1
    print(f"Análisis del scree plot: {inflection_point} factores")

    recommended_factors = min(kaiser_factors, factors_70)
    print(f"Número recomendado de factores: {recommended_factors}")

    # Esto es para tener 3 factores ya que originalmente solo habia 1
    manual_factors = 5
    print(f"Decision: Se usarán {manual_factors} factores en lugar de {recommended_

    return manual_factors

optimal_factors = determine_optimal_factors(ev, variance_proportion, cumulative_var
```

```
Criterio de Kaiser (eigenvalues > 1): 5 factores
Eigenvalues: ['8.697', '1.785', '1.443', '1.202', '1.073', '0.759', '0.745', '0.64
9', '0.644', '0.606']...
Varianza explicada:
Factor 1: 0.361 (0.361 acumulada)
Factor 2: 0.057 (0.419 acumulada)
Factor 3: 0.045 (0.464 acumulada)
Factor 4: 0.035 (0.498 acumulada)
Factor 5: 0.019 (0.517 acumulada)
Factor 6: 0.006 (0.523 acumulada)
Factor 7: 0.004 (0.528 acumulada)
Factor 8: 0.004 (0.532 acumulada)
Factor 9: 0.004 (0.535 acumulada)
Factor 10: 0.003 (0.538 acumulada)
Factores para 70% de varianza: 1
Factores para 80% de varianza: 1
Análisis del scree plot: 1 factores
Número recomendado de factores: 1
Decision: Se usarán 5 factores en lugar de 1 por business sense
```

In [48]:
```
def get_loadings_df(data: pd.DataFrame, rotation: str, optimal_factors: int):
    fa = FactorAnalyzer(n_factors=optimal_factors, rotation=rotation)
```

```
    fa.fit(data)

    loadings = fa.loadings_
    loadings_df = pd.DataFrame(
        loadings,
        index=satisfaction_vars,
        columns=[f'Factor_{i+1}' for i in range(optimal_factors)]
    )

    return loadings_df, fa
```

In [49]:
```
loadings_df_simple, fa_simple = get_loadings_df(data_scaled, None, optimal_factors)
print("Matriz de loadings de 3 factores (Sin rotación)")
print(loadings_df_simple.round(3))
```

```
Matriz de loadings de 3 factores (Sin rotación)
                          Factor_1  Factor_2  Factor_3  Factor_4  Factor_5
technical_expertise          0.711    -0.299    -0.155     0.146    -0.035
problem_solving              0.714    -0.323    -0.164     0.149    -0.039
innovation_solutions         0.713    -0.336    -0.154     0.165    -0.039
technical_documentation      0.718    -0.313    -0.135     0.146    -0.041
system_integration           0.721    -0.320    -0.135     0.150    -0.051
account_manager_responsive   0.614     0.001     0.353    -0.013    -0.084
executive_access             0.617     0.007     0.381     0.001    -0.060
trust_reliability            0.620     0.004     0.386    -0.019    -0.087
long_term_partnership        0.608    -0.007     0.385    -0.033    -0.053
communication_clarity        0.628     0.007     0.365    -0.011    -0.036
project_management           0.705     0.106    -0.157    -0.301     0.016
timeline_adherence           0.703     0.084    -0.158    -0.285     0.018
budget_control               0.705     0.083    -0.122    -0.278     0.044
quality_deliverables         0.700     0.086    -0.151    -0.290     0.008
change_management            0.696     0.084    -0.140    -0.283    -0.003
cost_transparency            0.440     0.378    -0.098     0.196    -0.008
value_for_money              0.435     0.340    -0.092     0.182    -0.009
roi_demonstration            0.434     0.379    -0.087     0.212    -0.018
competitive_pricing          0.429     0.379    -0.114     0.185    -0.051
billing_accuracy             0.442     0.415    -0.073     0.200    -0.085
support_responsiveness       0.367    -0.002     0.078     0.120     0.333
training_quality             0.357    -0.008     0.104     0.068     0.345
documentation_help           0.342    -0.004     0.072     0.059     0.332
```

```
c:\Users\crdie\anaconda3\envs\cienciaDatos\Lib\site-packages\sklearn\utils\deprecati
on.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in
1.8.
```

In [50]:
```
loadings_df_varimax, fa_varimax = get_loadings_df(data_scaled, "Varimax", optimal_f
print("Matriz de loadings de 3 factores (Varimax)")
print(loadings_df_varimax.round(3))
```

```
Matriz de loadings de 3 factores (Varimax)
                          Factor_1  Factor_2  Factor_3  Factor_4  Factor_5
technical_expertise          0.692     0.245     0.217     0.162     0.173
problem_solving              0.713     0.244     0.212     0.148     0.169
innovation_solutions         0.724     0.226     0.219     0.144     0.174
technical_documentation      0.698     0.240     0.238     0.150     0.173
system_integration           0.708     0.237     0.243     0.150     0.166
account_manager_responsive   0.217     0.193     0.612     0.172     0.145
executive_access             0.205     0.176     0.629     0.175     0.175
trust_reliability            0.205     0.190     0.643     0.166     0.148
long_term_partnership        0.195     0.195     0.629     0.143     0.175
communication_clarity        0.208     0.197     0.615     0.173     0.196
project_management           0.255     0.654     0.222     0.252     0.137
timeline_adherence           0.274     0.638     0.219     0.241     0.142
budget_control               0.261     0.622     0.243     0.233     0.173
quality_deliverables         0.269     0.637     0.226     0.238     0.132
change_management            0.269     0.626     0.236     0.238     0.123
cost_transparency            0.102     0.147     0.111     0.570     0.121
value_for_money              0.117     0.148     0.116     0.532     0.118
roi_demonstration            0.102     0.127     0.119     0.575     0.114
competitive_pricing          0.105     0.153     0.106     0.572     0.073
billing_accuracy             0.086     0.137     0.153     0.606     0.053
support_responsiveness       0.155     0.075     0.143     0.136     0.444
training_quality             0.123     0.102     0.160     0.096     0.449
documentation_help           0.123     0.113     0.130     0.098     0.427
```

```
c:\Users\crdie\anaconda3\envs\cienciaDatos\Lib\site-packages\sklearn\utils\deprecati
on.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in
1.8.
```

In [51]:
```
loadings_df_promax, fa_promax = get_loadings_df(data_scaled, "Promax", optimal_fact
print("Matriz de loadings de 3 factores (Varimax)")
print(loadings_df_promax.round(3))
```

```
c:\Users\crdie\anaconda3\envs\cienciaDatos\Lib\site-packages\sklearn\utils\deprecati
on.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in
1.8.
```

```
Matriz de loadings de 3 factores (Varimax)
                          Factor_1  Factor_2  Factor_3  Factor_4  Factor_5
technical_expertise          0.772     0.029    -0.003     0.013     0.007
problem_solving              0.806     0.025    -0.012    -0.004     0.002
innovation_solutions         0.824    -0.007    -0.000    -0.005     0.007
technical_documentation      0.778     0.017     0.028    -0.003     0.003
system_integration           0.794     0.009     0.035    -0.003    -0.009
account_manager_responsive   0.032     0.012     0.691     0.015    -0.024
executive_access             0.010    -0.018     0.714     0.018     0.015
trust_reliability            0.007     0.005     0.739     0.005    -0.023
long_term_partnership       -0.010     0.024     0.716    -0.027     0.018
communication_clarity        0.004     0.017     0.683     0.008     0.043
project_management          -0.001     0.780    -0.001     0.021    -0.003
timeline_adherence           0.033     0.751    -0.006     0.012     0.003
budget_control               0.008     0.725     0.027    -0.000     0.043
quality_deliverables         0.026     0.753     0.008     0.010    -0.010
change_management            0.030     0.734     0.028     0.013    -0.023
cost_transparency           -0.006     0.012    -0.019     0.607     0.035
value_for_money              0.016     0.018    -0.011     0.559     0.032
roi_demonstration            0.001    -0.020    -0.002     0.620     0.026
competitive_pricing          0.008     0.026    -0.017     0.616    -0.027
billing_accuracy            -0.022    -0.010     0.060     0.660    -0.063
support_responsiveness       0.041    -0.046     0.002     0.039     0.496
training_quality            -0.017     0.011     0.030    -0.021     0.506
documentation_help          -0.009     0.036    -0.008    -0.013     0.483
```

Al analizar los valores con rotación y sin rotación se puede observar como se logra una mayor interpretabilidad al usar la rotaciones, esto debido a que cuando no se realiza se puede perder cual es el conjunto de variables que mejor describen al factor. Por otro lado la rotacion promax ayuda a una mayor interpretabilidad de los datos.

In [52]:
```python
# interpretacion de factores (corregido por Deepseek)
def interpret_factors(loadings_df, threshold=0.5):
    print(f"Umbral para loadings significativos: |loading| ≥ {threshold}")

    factor_interpretations = {}

    for factor in loadings_df.columns:
        print(f"{factor}")

        # Variables con loadings altos en este factor
        high_loadings = loadings_df[abs(loadings_df[factor]) >= threshold][factor]
        high_loadings = high_loadings.sort_values(ascending=False)

        print("Variables con loadings significativos:")
        for var, loading in high_loadings.items():
            print(f"  {var}: {loading:.3f}")

        # Interpretar el factor basado en las variables
        positive_vars = [var for var, loading in high_loadings.items() if loading >

        if positive_vars:
            # Agrupar por dimensiones teóricas
            tech_count = sum(1 for var in positive_vars if var in technical_vars)
            rel_count = sum(1 for var in positive_vars if var in relationship_vars)
```

```
            proj_count = sum(1 for var in positive_vars if var in project_vars)
            val_count = sum(1 for var in positive_vars if var in value_vars)
            supp_count = sum(1 for var in positive_vars if var in support_vars)

            counts = {
                'Técnico': tech_count,
                'Relación': rel_count,
                'Proyecto': proj_count,
                'Valor': val_count,
                'Soporte': supp_count
            }

            primary_domain = max(counts, key=counts.get)

            print(f"Dominio principal: {primary_domain}")
            print(f"Distribución: {counts}")

            factor_interpretations[factor] = {
                'primary_domain': primary_domain,
                'variables': positive_vars,
                'domain_distribution': counts
            }

    return factor_interpretations

#interpretar factores
print("Ya se muestran en la grafica anterior, pero para tenerlos más a la mano: ")
factor_interpretations = interpret_factors(loadings_df_promax, threshold=0.4)
```

```
Ya se muestran en la grafica anterior, pero para tenerlos más a la mano:
Umbral para loadings significativos: |loading| ≥ 0.4
Factor_1
Variables con loadings significativos:
  innovation_solutions: 0.824
  problem_solving: 0.806
  system_integration: 0.794
  technical_documentation: 0.778
  technical_expertise: 0.772
Dominio principal: Técnico
Distribución: {'Técnico': 5, 'Relación': 0, 'Proyecto': 0, 'Valor': 0, 'Soporte': 0}
Factor_2
Variables con loadings significativos:
  project_management: 0.780
  quality_deliverables: 0.753
  timeline_adherence: 0.751
  change_management: 0.734
  budget_control: 0.725
Dominio principal: Proyecto
Distribución: {'Técnico': 0, 'Relación': 0, 'Proyecto': 5, 'Valor': 0, 'Soporte': 0}
Factor_3
Variables con loadings significativos:
  trust_reliability: 0.739
  long_term_partnership: 0.716
  executive_access: 0.714
  account_manager_responsive: 0.691
  communication_clarity: 0.683
Dominio principal: Relación
Distribución: {'Técnico': 0, 'Relación': 5, 'Proyecto': 0, 'Valor': 0, 'Soporte': 0}
Factor_4
Variables con loadings significativos:
  billing_accuracy: 0.660
  roi_demonstration: 0.620
  competitive_pricing: 0.616
  cost_transparency: 0.607
  value_for_money: 0.559
Dominio principal: Valor
Distribución: {'Técnico': 0, 'Relación': 0, 'Proyecto': 0, 'Valor': 5, 'Soporte': 0}
Factor_5
Variables con loadings significativos:
  training_quality: 0.506
  support_responsiveness: 0.496
  documentation_help: 0.483
Dominio principal: Soporte
Distribución: {'Técnico': 0, 'Relación': 0, 'Proyecto': 0, 'Valor': 0, 'Soporte': 3}
```

In [53]:

```python
# Calidad de la solución factorial (Ayudado por deepseek)
def assess_factor_solution_quality(loadings_df, threshold=0.5):
    # Porcentaje de varianza explicada por cada factor
    fa_final = FactorAnalyzer(n_factors=len(loadings_df.columns), rotation='varimax
    fa_final.fit(data_scaled)
    variance_info = fa_final.get_factor_variance()

    print("Varianza explicada por cada factor:")
    for i, (var, prop, cum) in enumerate(zip(variance_info[0], variance_info[1], va
        print(f"{loadings_df.columns[i]}: {var:.3f} ({prop:.3f} proporción)")
```

```
    print(f"Total: {sum(variance_info[0]):.3f} ({sum(variance_info[1]):.3f} proporc

    print(f"Estructura simple (loadings ≥ {threshold}):")
    simple_structure = (abs(loadings_df) >= threshold).sum(axis=1)
    cross_loadings = sum(simple_structure > 1)
    no_loading = sum(simple_structure == 0)

    print(f"Variables con cross loadings: {cross_loadings}")
    print(f"Variables sin loadings significativos: {no_loading}")
    print(f"Variables con loading claro: {len(loadings_df) - cross_loadings - no_lo

    # Comunalidades
    communalities = fa_final.get_communalities()
    comm_df = pd.DataFrame({
        'Variable': satisfaction_vars,
        'Comunalidad': communalities
    }).sort_values('Comunalidad')

    print(f"Comunalidades (rango: {communalities.min():.3f} - {communalities.max():
    print("Variables con menor comunalidad:")
    print(comm_df.head().to_string(index=False))

assess_factor_solution_quality(loadings_df_promax)
```

```
Varianza explicada por cada factor:
Factor_1: 3.173 (0.138 proporción)
Factor_2: 2.615 (0.114 proporción)
Factor_3: 2.613 (0.114 proporción)
Factor_4: 2.212 (0.096 proporción)
Factor_5: 1.021 (0.044 proporción)
Total: 11.633 (0.506 proporción)
Estructura simple (loadings ≥ 0.5):
Variables con cross loadings: 0
Variables sin loadings significativos: 2
Variables con loading claro: 21
Comunalidades (rango: 0.236 - 0.674)
Variables con menor comunalidad:
              Variable  Comunalidad
    documentation_help     0.236420
      training_quality     0.261606
support_responsiveness     0.265861
       value_for_money     0.346037
   competitive_pricing     0.377859
```

```
c:\Users\crdie\anaconda3\envs\cienciaDatos\Lib\site-packages\sklearn\utils\deprecati
on.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in
1.8.
```

Se realizaron pruebas cambiando el numero de factores entre 1 y 5, que fueron los rangos que nos dió el criterio de Kaiser y el mínimo para alcanzar una varianza explicada del 70% al inicio, se decidió usar 5 factores ya que estos coincidian perfectamente con las 5 dimensiones generales.

Esta solución de análisis por factores logra tener 0 variables con cross loadings, lo que permite que los factores son lo suficientemente distintos entre sí, además de que 21 variables se agrupan claramente, dejando solamente a 2 sin agruparse, lo cual es algo bastante bueno. Cada uno de los factores representa muy bien las 5 dimensiones del area de satisfacción al cliente. Cada factor se podría renombrar de acuerdo con las variables que explican de la siguiente manera:

- Factor_1 : technical_excellence
- Factor_2 : project_delivery
- Factor_3 : relationship_management
- Factor_4 : value_cost
- Factor_5 : support

In [54]:
```
#Calculo de factor scores para cada usuario
factor_scores = fa_promax.transform(data_scaled)

names = {'Factor_1': 'technical_excellence', 'Factor_2': 'project_delivery', 'Facto

factor_scores_df = pd.DataFrame(
    factor_scores,
    columns=[names[f'Factor_{i+1}'] for i in range(optimal_factors)],
    index=data_scaled.index
)

factor_scores_df
```

```
c:\Users\crdie\anaconda3\envs\cienciaDatos\Lib\site-packages\sklearn\utils\deprecati
on.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in
1.8.
```

Out[54]:

customer_id	technical_excellence	project_delivery	relationship_management	value_cost	s
CUST_001	0.572905	0.484987	1.072690	1.546534	0
CUST_002	-0.303997	0.045093	0.846662	-0.767469	-0
CUST_003	-0.843954	-0.636480	-0.308274	-0.917284	-0
CUST_004	-0.691342	-0.712976	-1.784196	-0.817435	-1
CUST_005	1.007371	0.839535	1.109080	0.161517	0
...	...	...	...	...	
CUST_846	1.157605	0.423766	0.285101	0.461150	0
CUST_847	-0.539915	-0.166255	-0.160987	0.068587	-0
CUST_848	1.429937	1.626330	0.146201	-0.081155	0
CUST_849	-0.623396	-0.585522	-0.675635	-0.013841	-0
CUST_850	-0.027892	-0.994264	-1.035963	-0.290101	-0

3400 rows × 5 columns

Determinación de factores clave

In [55]:
```
#Prediciendo a las variables, se realizará un modelo independiente para cada variab

#Se probaron varios modelos (Linear Regression, DT y RF) y Random Forest fue el que
X = factor_scores_df
y_s = data_imputed[outcome_vars]

general_importances = {}

for y in y_s:
    print(f'\nEntrenamiento de modelo para outcome: {y}')
    X_train, X_test, y_train, y_test = train_test_split(X, y_s[y], test_size=0.2, r
    model = RandomForestRegressor(max_depth=30, n_estimators=100, max_leaf_nodes=20
    model.fit(X_train, y_train)
    print(f"R-squared (precisión) en test: {model.score(X_test, y_test):.4f}")
    y_pred = model.predict(X_test)
    print(f"MSE en test: {mean_squared_error(y_test, y_pred):.4f}")

    #Obteniendo los factores releavantes por cada modelo
    importances = model.feature_importances_
    importance_df = pd.DataFrame({
        'Factor': X.columns,
        'Importance': importances
    }).sort_values(by='Importance', ascending=False)
```

```
    general_importances[y] = importance_df
    print(importance_df)
```

```
Entrenamiento de modelo para outcome: overall_satisfaction
R-squared (precisión) en test: 0.5960
MSE en test: 0.2597
                   Factor  Importance
1         project_delivery    0.369976
0     technical_excellence    0.261993
4                  support    0.235746
3               value_cost    0.077647
2  relationship_management    0.054638

Entrenamiento de modelo para outcome: nps_score
R-squared (precisión) en test: 0.2360
MSE en test: 2.4204
                   Factor  Importance
4                  support    0.455313
1         project_delivery    0.272141
2  relationship_management    0.124744
3               value_cost    0.089026
0     technical_excellence    0.058775

Entrenamiento de modelo para outcome: renewal_likelihood
R-squared (precisión) en test: 0.3622
MSE en test: 0.4069
                   Factor  Importance
1         project_delivery    0.477282
0     technical_excellence    0.189141
4                  support    0.130177
2  relationship_management    0.128116
3               value_cost    0.075285

Entrenamiento de modelo para outcome: revenue_growth_pct
R-squared (precisión) en test: 0.5451
MSE en test: 28.2944
                   Factor  Importance
1         project_delivery    0.284625
2  relationship_management    0.256017
0     technical_excellence    0.252988
4                  support    0.143211
3               value_cost    0.063159

Entrenamiento de modelo para outcome: referrals_generated
R-squared (precisión) en test: 0.2453
MSE en test: 1.7691
                   Factor  Importance
1         project_delivery    0.415010
4                  support    0.168279
2  relationship_management    0.167051
0     technical_excellence    0.162463
3               value_cost    0.087197
```

Cada outcome variable tiene ciertos factores que son mas relevantes que otros, esto se puede ver muy claramente en el output anterior, en el que se muestra el orden de importancia de cada factor en cada outcome.

Asumiendo que para el negocio las 5 outcome variables son igualmente de importantes, se calculará el orden de importancia de cada factor de acuerdo a la suma de su importancia en cada variable outcome de manera individual.

En el caso que alguan variable tuviera mas importancia que otra se le pueden asignar pesos para ajustar los diferentes intereses de la empresa.

In [56]:
```
combined_importances_df = pd.concat(general_importances.values())

total_importances = combined_importances_df.groupby('Factor')['Importance'].sum()

sorted_importances = total_importances.sort_values(ascending=False)

print(sorted_importances)
```

```
Factor
project_delivery           1.819034
support                    1.132725
technical_excellence       0.925359
relationship_management    0.730567
value_cost                 0.392314
Name: Importance, dtype: float64
```

In [57]:
```
#Importancia relativa
normalized_importances = (sorted_importances / sorted_importances.sum()) * 100

print(normalized_importances.round(2).astype(str) + ' %')
```

```
Factor
project_delivery           36.38 %
support                    22.65 %
technical_excellence       18.51 %
relationship_management    14.61 %
value_cost                  7.85 %
Name: Importance, dtype: object
```

In [58]:
```
print("Definición del Factor 'project_delivery (Factor_2)':")
print(loadings_df_promax.sort_values(by='Factor_2', ascending=False).head(3))
```

```
Definición del Factor 'project_delivery (Factor_2)':
                     Factor_1  Factor_2  Factor_3  Factor_4  Factor_5
project_management  -0.000633  0.780053 -0.001090  0.020874 -0.002851
quality_deliverables  0.026141  0.752915  0.008436  0.009724 -0.010246
timeline_adherence    0.033041  0.751470 -0.006363  0.012484  0.003314
```

In [59]:
```
print("Definición del Factor 'support (Factor_5)':")
print(loadings_df_promax.sort_values(by='Factor_5', ascending=False).head(3))
```

```
Definición del Factor 'support (Factor_5)':
                       Factor_1  Factor_2  Factor_3  Factor_4  Factor_5
training_quality      -0.016911  0.010611  0.030372 -0.020523  0.506189
support_responsiveness  0.041109 -0.046486  0.001631  0.039384  0.495715
documentation_help    -0.009125  0.036011 -0.008367 -0.013153  0.482867
```

In [60]:
```
print("Clientes con peor percepción de 'relationship_management':")
at_risk_customers_value = factor_scores_df.sort_values(by='relationship_management'

print(at_risk_customers_value.head(30))
```

```
Clientes con peor percepción de 'relationship_management':
             technical_excellence  project_delivery  relationship_management  \
customer_id
CUST_455                -3.405342         -2.856138                -3.014233
CUST_577                -2.766397         -2.584414                -2.962637
CUST_037                -2.699126         -2.434930                -2.896264
CUST_810                -1.479393         -1.976020                -2.895325
CUST_672                -2.098898         -2.839582                -2.725259
CUST_386                -1.200669         -1.617480                -2.721116
CUST_166                -1.508631         -0.892597                -2.691315
CUST_569                -2.984211         -3.088916                -2.638372
CUST_689                -2.091629         -2.237322                -2.607614
CUST_671                -0.398798         -1.541861                -2.594433
CUST_543                -3.166359         -2.647593                -2.592999
CUST_817                -0.513360         -1.225679                -2.590234
CUST_364                -0.789351         -1.636510                -2.533777
CUST_756                -1.693017         -2.247040                -2.391672
CUST_231                -1.257597         -2.296482                -2.388623
CUST_319                -1.221420         -1.828196                -2.347183
CUST_270                -1.811745         -1.341901                -2.338614
CUST_722                -1.607901         -2.038605                -2.330388
CUST_565                -1.379350         -1.678092                -2.323016
CUST_222                -0.839683          0.267337                -2.322576
CUST_607                -2.686171         -2.209614                -2.295483
CUST_456                -1.201913         -1.608324                -2.295398
CUST_443                -1.159600         -1.500591                -2.282155
CUST_333                -2.627863         -2.041979                -2.276660
CUST_525                -2.480026         -2.120534                -2.270107
CUST_479                -1.924418         -2.700113                -2.258632
CUST_632                -1.381568         -1.384498                -2.246437
CUST_149                -1.363396         -1.232481                -2.243481
CUST_362                -1.355221         -1.259257                -2.232108
CUST_376                -1.653397         -2.682672                -2.228006

             value_cost   support
customer_id
CUST_455      -1.204018 -2.698506
CUST_577      -0.715811 -2.879343
CUST_037      -1.893180 -1.663962
CUST_810      -2.454883 -2.862081
CUST_672      -2.868861 -1.521895
CUST_386      -0.797936 -1.743834
CUST_166      -1.470223 -1.011704
CUST_569      -2.619933 -2.675619
CUST_689      -0.651591 -1.607885
CUST_671      -0.726978 -1.247387
CUST_543      -1.613995 -2.582721
CUST_817      -0.872788 -1.271006
CUST_364      -1.440987 -1.831554
CUST_756      -1.279089 -1.791953
CUST_231      -2.307632 -1.363030
CUST_319      -0.989760 -1.922941
CUST_270      -1.747321 -2.110143
CUST_722      -0.777337 -0.788220
CUST_565      -2.036345 -0.987056
CUST_222      -0.640395 -0.975609
```

```
CUST_607      -2.184505 -1.843165
CUST_456      -0.918898 -1.337350
CUST_443      -2.239230 -1.204747
CUST_333      -2.620751 -1.454562
CUST_525      -1.064305 -2.206832
CUST_479      -1.805589 -1.876297
CUST_632      -0.269527 -1.578066
CUST_149      -1.050348 -1.446349
CUST_362      -1.469619 -0.885136
CUST_376      -1.619394 -1.382548
```

In [61]:
```python
print("Clientes con peor percepción de 'support':")
at_risk_customers_value = factor_scores_df.sort_values(by='support', ascending=True

print(at_risk_customers_value.head(30))
```

```
Clientes con peor percepción de 'support':
             technical_excellence  project_delivery  relationship_management  \
customer_id
CUST_577                -2.766397         -2.584414                -2.962637
CUST_810                -1.479393         -1.976020                -2.895325
CUST_455                -3.405342         -2.856138                -3.014233
CUST_569                -2.984211         -3.088916                -2.638372
CUST_543                -3.166359         -2.647593                -2.592999
CUST_278                -0.603586         -1.858506                -1.261264
CUST_258                -1.622716         -1.514257                -2.180949
CUST_525                -2.480026         -2.120534                -2.270107
CUST_577                -2.135402         -0.835781                -1.120463
CUST_850                -1.579468         -1.462260                -2.004085
CUST_117                -1.794566         -1.415556                -1.176321
CUST_470                -1.870802         -2.628279                -2.092894
CUST_761                -1.427035         -2.067616                -1.799026
CUST_270                -1.811745         -1.341901                -2.338614
CUST_308                -1.933702         -1.190514                -0.284437
CUST_619                -0.899304         -1.257697                -0.447962
CUST_398                -2.030513         -2.122630                -2.051542
CUST_163                -2.405789         -1.312263                -1.929998
CUST_177                -2.635828         -2.099545                -1.063798
CUST_420                -2.173518         -1.114483                -2.207137
CUST_114                -0.759202         -1.281324                -2.006738
CUST_473                -0.999074         -2.117113                -1.377528
CUST_319                -1.221420         -1.828196                -2.347183
CUST_619                -0.740693         -0.850091                -1.991537
CUST_673                -1.217544         -2.157757                -2.180858
CUST_180                -1.343857         -0.773710                -1.822296
CUST_356                -0.721012         -0.962227                -1.620101
CUST_271                -1.216300         -1.607214                -2.100938
CUST_623                -1.401979         -1.835919                -1.757721
CUST_516                -1.395413         -1.865536                -1.577928

             value_cost   support
customer_id
CUST_577      -0.715811 -2.879343
CUST_810      -2.454883 -2.862081
CUST_455      -1.204018 -2.698506
CUST_569      -2.619933 -2.675619
CUST_543      -1.613995 -2.582721
CUST_278      -0.247830 -2.288866
CUST_258      -1.841930 -2.213673
CUST_525      -1.064305 -2.206832
CUST_577      -1.082716 -2.178984
CUST_850      -1.819569 -2.172917
CUST_117      -1.330396 -2.156137
CUST_470      -3.072670 -2.150435
CUST_761      -1.876898 -2.149954
CUST_270      -1.747321 -2.110143
CUST_308      -1.229574 -2.104875
CUST_619      -0.427022 -2.068163
CUST_398      -1.922792 -2.044108
CUST_163      -1.348350 -2.029027
CUST_177      -2.099344 -2.003523
CUST_420      -1.517440 -1.982127
```

```
CUST_114      -2.906344 -1.952972
CUST_473      -1.280425 -1.938394
CUST_319      -0.989760 -1.922941
CUST_619       0.499161 -1.903280
CUST_673      -2.978613 -1.899853
CUST_180      -0.345514 -1.898943
CUST_356      -1.034695 -1.892308
CUST_271      -1.123740 -1.889595
CUST_623      -1.610143 -1.887690
CUST_516      -2.248790 -1.878103
```

Visualizaciones de apoyo

In [62]:
```python
#Mejora de codigo con Gemini para visualizaciones claras y mas profesionales

fig = px.imshow(
    loadings_df_promax,
    text_auto='.2f',  # Muestra los valores numéricos con 2 decimales
    aspect="auto",
    color_continuous_scale='RdBu_r', # Escala de Rojo a Azul (centrada en 0)
    zmin=-1, zmax=1,  # Fija la escala de -1 a 1
    title="<b>Composición de los Factores (Factor Loadings)</b>"
)

fig.update_layout(
    xaxis_title="Factores Identificados",
    yaxis_title="Métricas de Satisfacción (Variables)",
    width=800,
    height=1000,
    title_x=0.5
)

fig.show()
```

In [63]:
```python
#Mejora de codigo con Gemini para visualizaciones claras y mas profesionales
df_melted = factor_scores_df.melt(var_name='Factor', value_name='Score')

fig = px.violin(
    df_melted,
    y='Score',
    x='Factor',
    box=True,  # Muestra el box plot adentro
    points=False, # Oculta los puntos individuales
    title='<b>Distribución de Puntuaciones de Clientes por Factor</b>'
)

fig.update_layout(
    xaxis_title="Factor",
    yaxis_title="Puntuación (Score)",
    title_x=0.5
)

fig.show()
```

In [64]:
```
#Mejora de codigo con Gemini para visualizaciones claras y mas profesionales

plot_df = pd.concat(general_importances, names=['Outcome'])

# 2. Resetea el índice para que 'Outcome' sea una columna estándar
plot_df = plot_df.reset_index()

# 3. Crea el gráfico de barras apiladas
fig = px.bar(
    plot_df,
    x='Outcome',          # Cada barra es un outcome
    y='Importance',       # El valor a apilar
    color='Factor',       # Los segmentos de color son los factores
    text_auto='.1%',      # Muestra el valor como porcentaje con 1 decimal
    title='<b>Desglose del Impacto: Qué Factor Impulsa Cada Resultado de Negocio</b
)

# 4. Ajusta el layout para claridad
fig.update_layout(
    yaxis_title="Contribución de Importancia (Total 100%)",
    xaxis_title="Variable de Resultado (Outcome)",
    legend_title="Factores",
    title_x=0.5,
    height = 600
)

fig.show()
```

In [65]:
```
#Mejora de codigo con Gemini para visualizaciones claras y mas profesionales

if isinstance(normalized_importances, pd.Series):
    df_impact = normalized_importances.reset_index()
    df_impact.columns = ['Factor', 'Importancia (%)']
else:
    df_impact = normalized_importances

fig = px.bar(
    # Ordena de menor a mayor para que el gráfico se muestre de arriba hacia abajo
    df_impact.sort_values(by='Importancia (%)', ascending=True),
    x='Importancia (%)',
    y='Factor',
    orientation='h',
    text='Importancia (%)',
    title='<b>Resumen de Impacto: Importancia Agregada de Cada Factor</b>'
)

# Formatea el texto para que muestre el %
fig.update_traces(texttemplate='%{text:.1f}%', textposition='outside')
fig.update_layout(
    xaxis_title="Impacto Total en Resultados de Negocio (%)",
    yaxis_title="Factor",
    title_x=0.5
)
```

```
fig.show()
```